先端エレクトロニクスDAQセミナー2020

FPGAを用いたDAQ技術概要

濱田 英太郎

KEK 素核研

自己紹介

- 名前：濱田　英太郎
- 所属：KEK　素粒子原子核研究所
- 職位：准技師
- 経歴：

  学生の時はBNL PHENIX実験のメンバー

  卒業後、IT企業に就職

  2013年からKEKに技術職員として勤務
- 仕事内容

  主にFPGAファームウェア、ASIC（デジタル部）、DAQソフトウェアの開発

目的と目標

- 目的

  FPGAがどのようなものか、どのようにして開発を進めるのか、どのようにDAQに使えるのかの概要を知る。

- 目標

  今後の研究開発にFPGAを用いるきっかけを作る。

目次

1. FPGAの仕組み
   - FPGAとは
   - FPGAの基本構造
   - ザイリンクス社FPGAの構成
2. FPGAファームウェア開発
   - 開発の流れ
   - 設計手法
3. FPGAを用いたDAQ
   - 一般的なDAQ回路
   - COMET実験ストロー飛跡検出器用読み出し回路におけるFPGA
4. 最後に

1. FPGAの仕組み

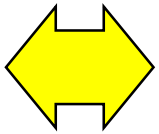
FPGA（Field Programmable Gate Array）とは

- プログラム可能なLSIの1種
  - 主としてデジタル（論理）回路
- 回路情報をダウンロードして論理回路を実現
  - あらかじめ論理素子が配置されている
  - 何度でも簡単に書き換えが可能
- Xilinx社とIntel社（元Altera社）の2社で8割超のシェア

※この講義では、現在の主流のFPGAであるSRAM型FPGAについて話します。
SRAM型FPGAでは何度でも簡単に書き換えが可能です。

FPGA実装の長所と短所

- 長所
  - 簡単にICを作れる
  - 回路修正が容易



- ソフトウエアの様にダウンロードして使用する
  - 時と場合により機能変更できる
  - 実験しながら処理方法を決定できる
- 高い柔軟性
- ハードウエアの欠点である低い柔軟性を克服する

- 短所
  - デジタルのみ
  - 単価が他の市販汎用デジタル集積回路と比較すると多少高価

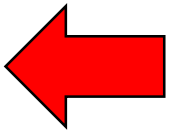
- 私たちが使用する場合はASICと比較しても性能が見劣りしなくなっている。

その理由は？

製造半導体プロセスルール

量産可能な最先端の半導体プロセスを使用している

0.18μm, 0.13μm, 90nm, 65nm, 40nm, 28nm, 20nm, 16nm



私たちが使えるFPGA技術はこの辺り

KEKで使用してきたアナログデジタル混在センサー信号処理用集積回路開発プロセス

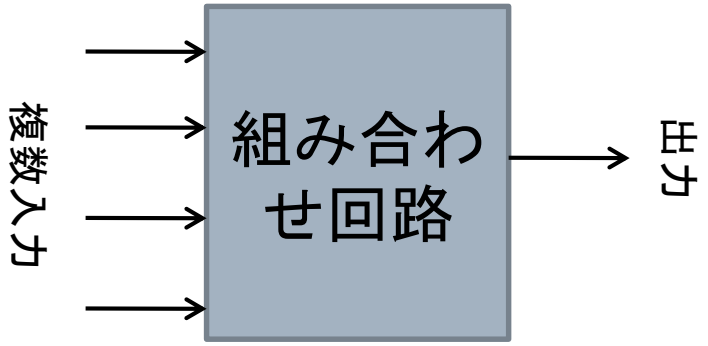
FPGAを使うことで3世代先の技術を使えるため、
FPGAの欠点を補う効果がある

FPGAの基本構造の前に…

デジタル回路は組み合わせ回路と順序回路に分類される

組み合わせ回路

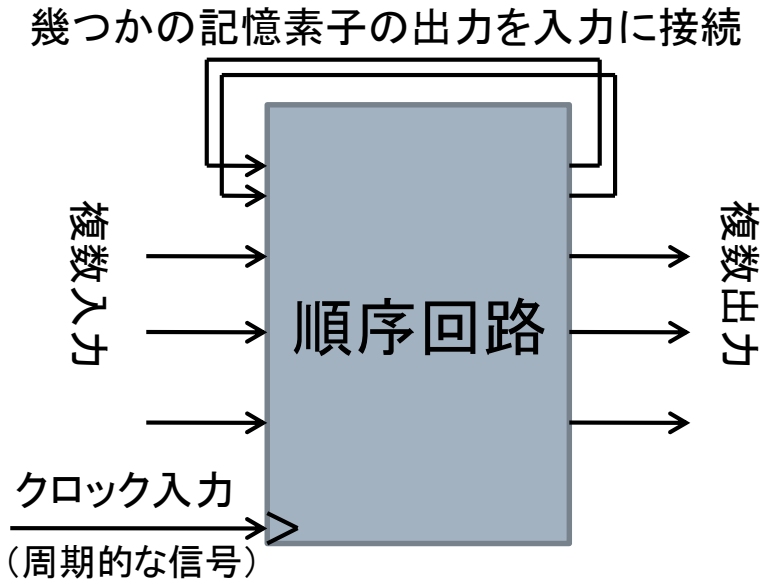
記憶素子を含まない回路



- 入出力の対応が1:1で決まっている
- 入出力を真理値表で書ける

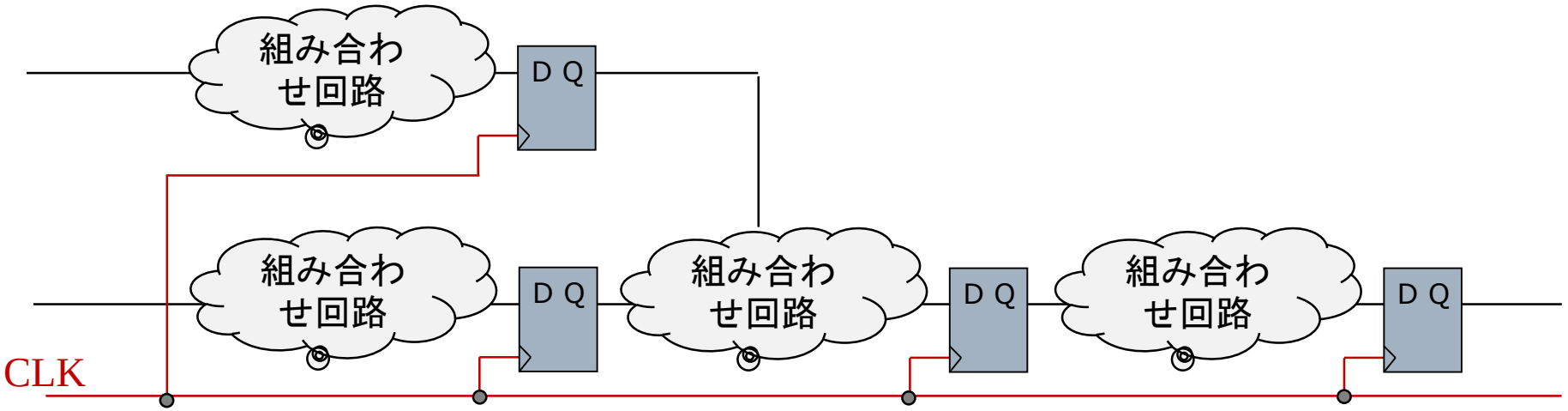
順序回路

記憶素子を含む回路



FPGAの基本構造の前に…

デジタル回路は組み合わせ回路と記憶素子で回路を作成

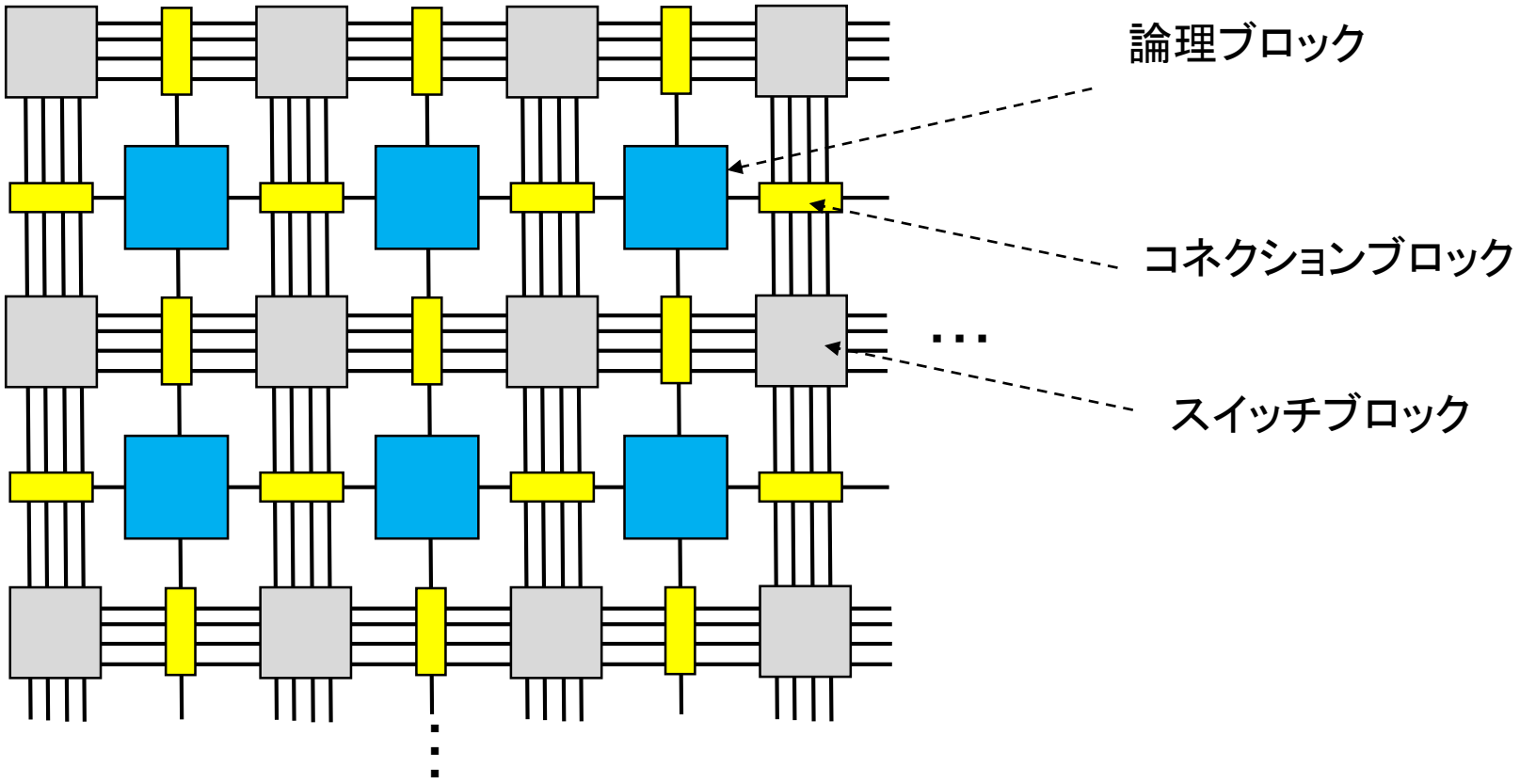


組み合わせ回路と記憶素子で作成されたデジタル回路

FPGAの構成　→　自由に変更可能な組み合わせ回路と記憶素子＋
それを自由に配線できる

FPGAの基本構造

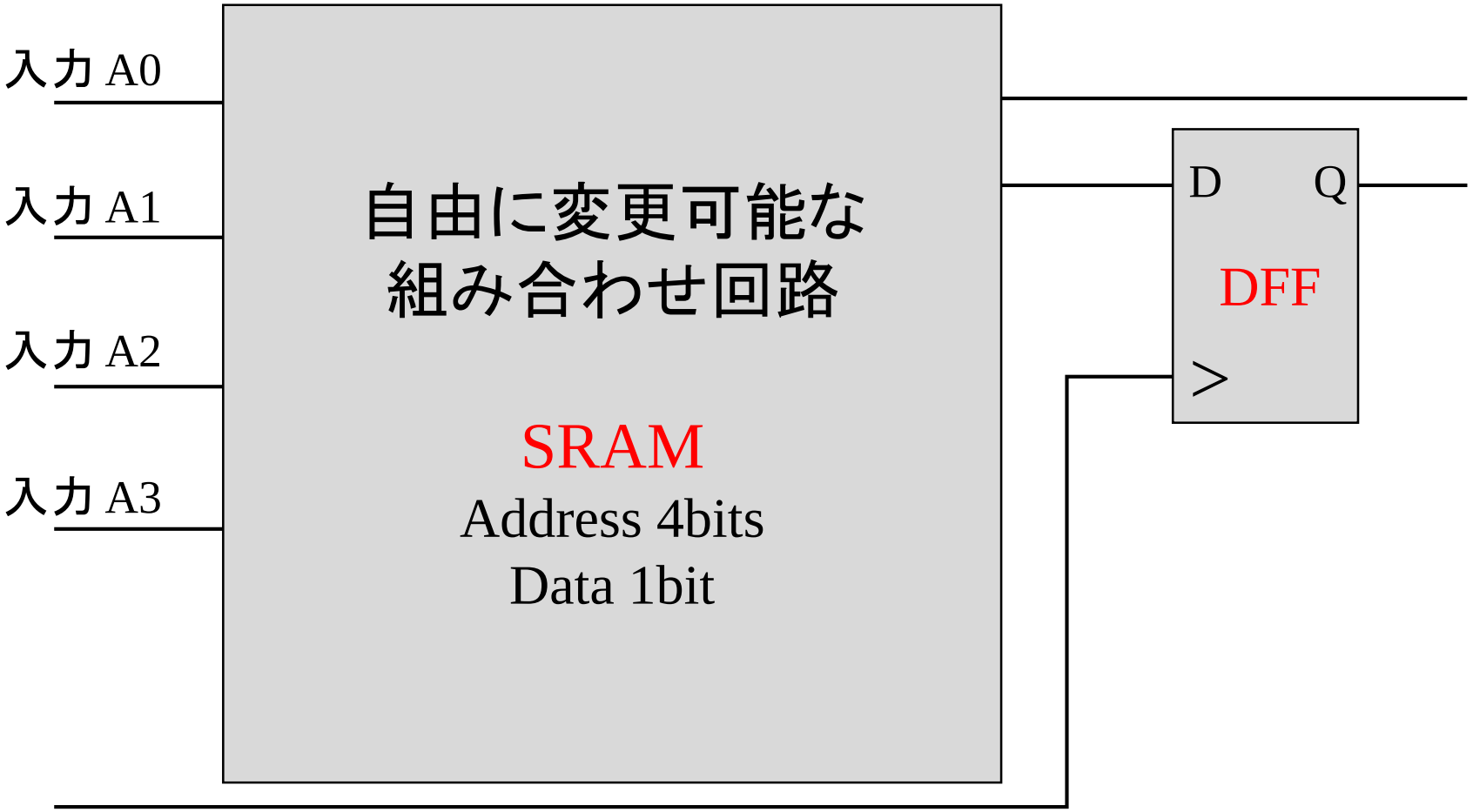
基本構造は論理ブロックと
配線要素（配線、スイッチブロック、コネクションブロック）



FPGAの基本構造

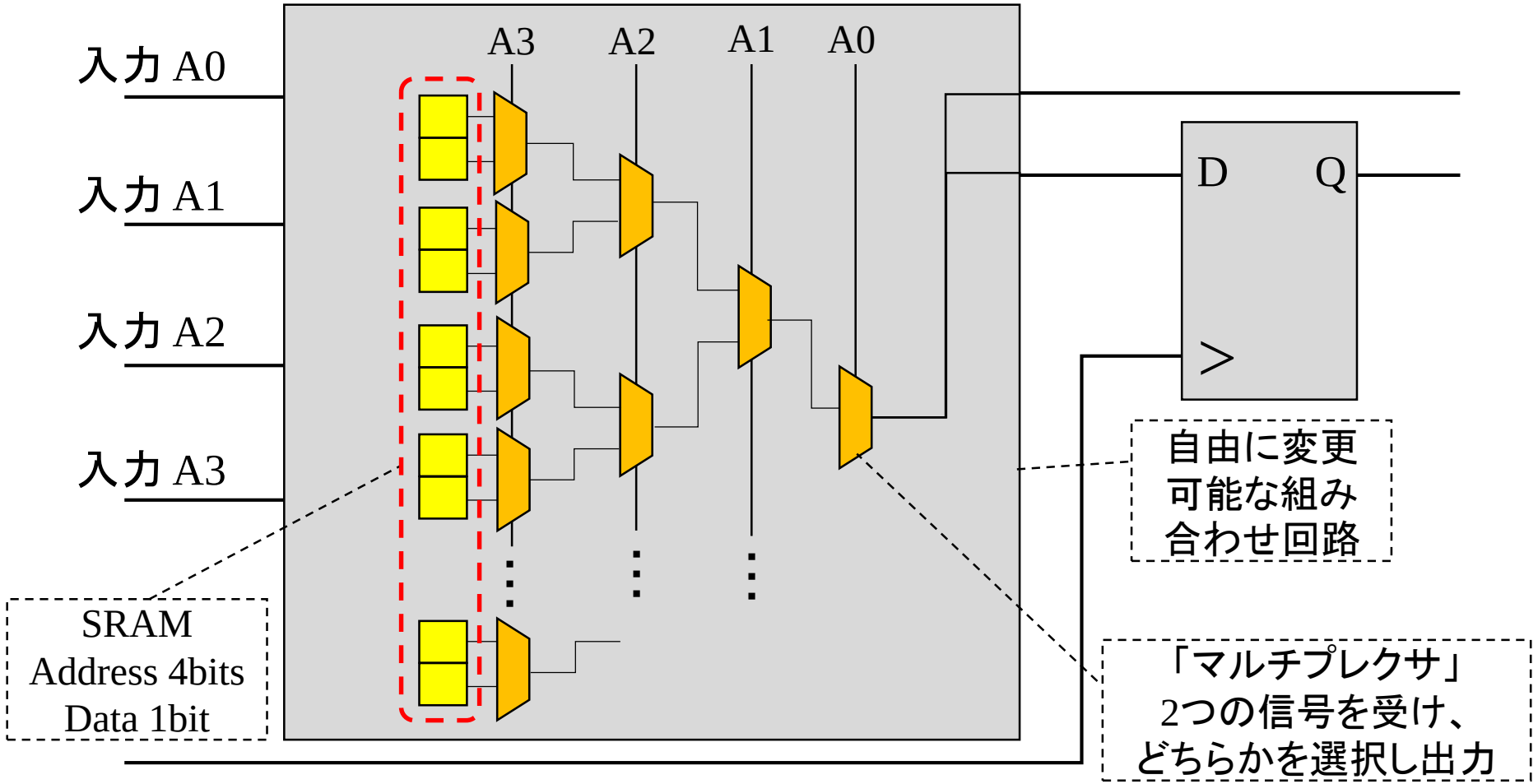
論理ブロック

論理ブロックの基本はSRAM + DFF



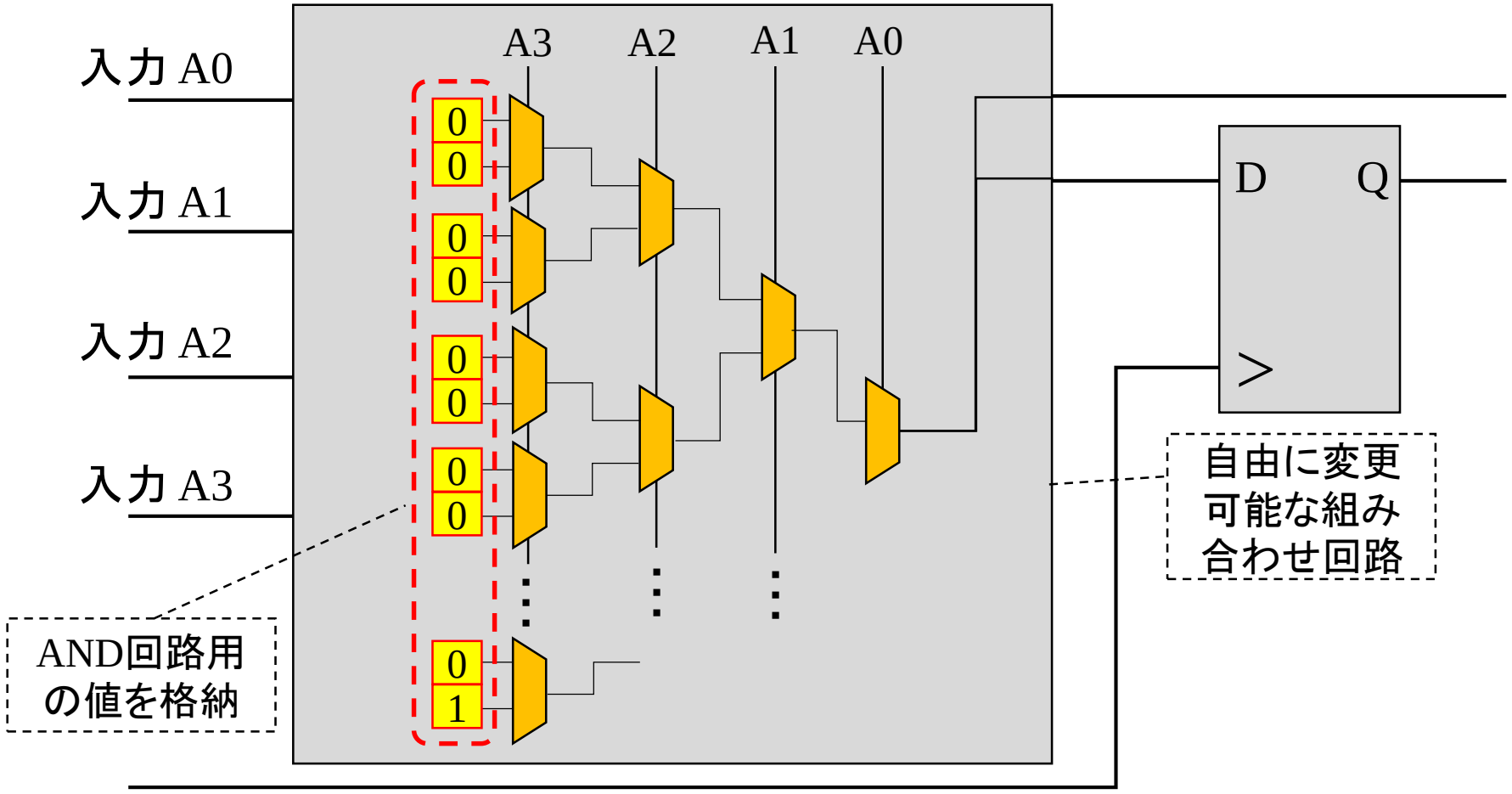
論理ブロック

論理ブロックの基本はSRAM + DFF



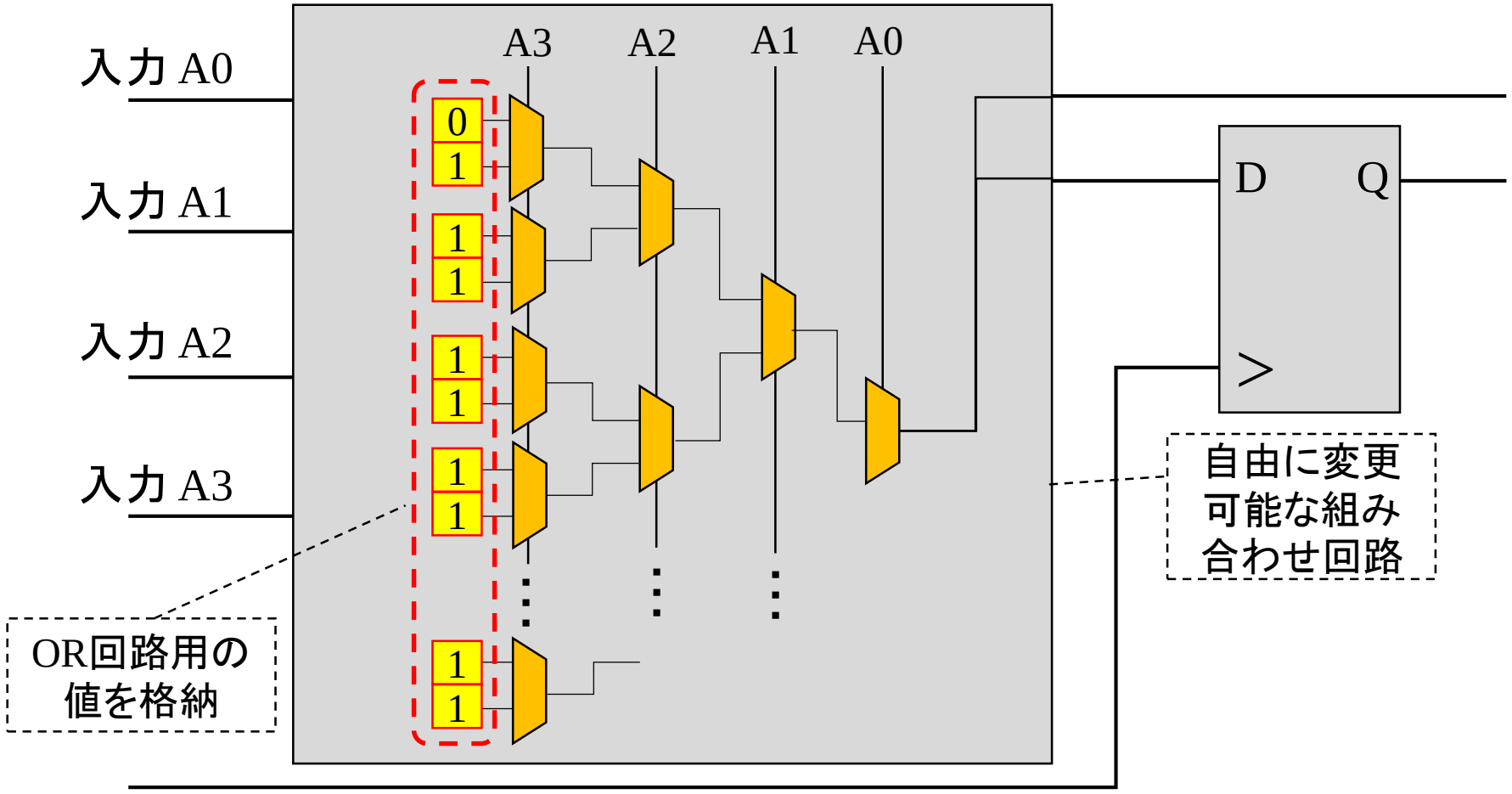
４入力ANDの例

論理ブロックの基本はSRAM + DFF

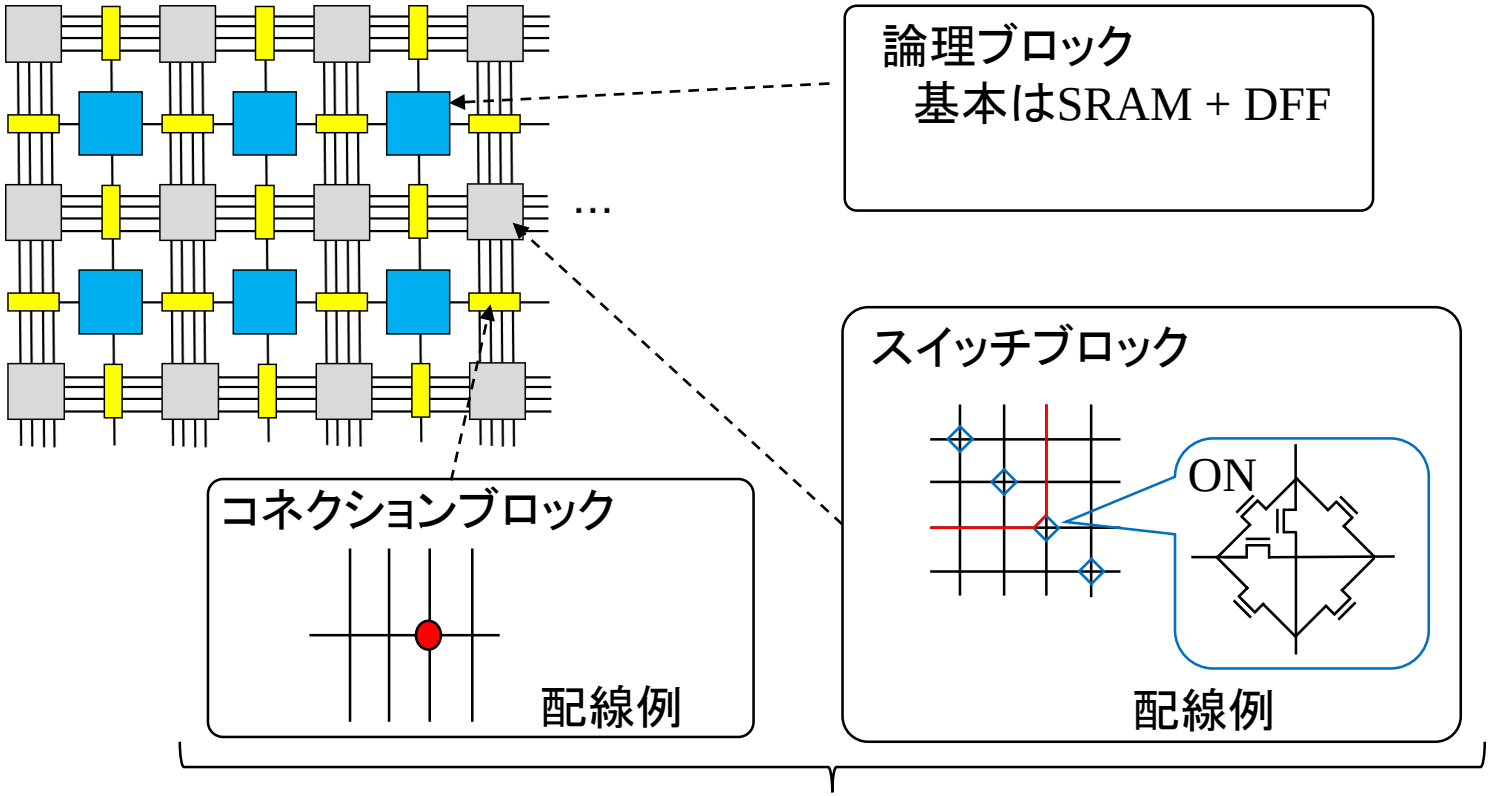


４入力ORの例

論理ブロックの基本はSRAM + DFF



FPGAの基本構造



トランジスタのON/OFFで接続を決める
（コンフィグレーションメモリで決める）ことで、
論理ブロックの配線を自由に行うことが可能

ザイリンクス社FPGAの場合

- 論理ブロックのSRAMはLUT (Look Up Table)と言う

- 論理ブロックはCLB（Configuration Logic Block）と言う
  - CLBには4つのSliceがある
  - SliceはLUT+DFF＋αで構成

Sliceの構造(7 series)

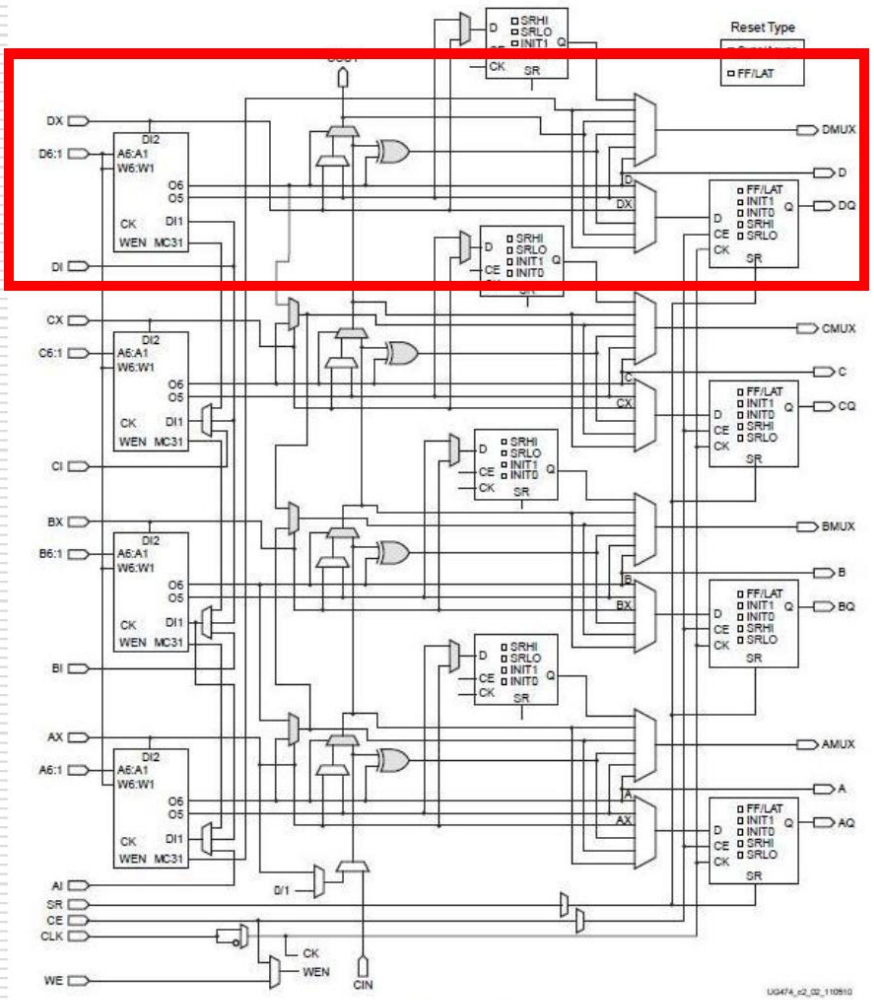


Figure 2-2: **Diagram of SLICEM**

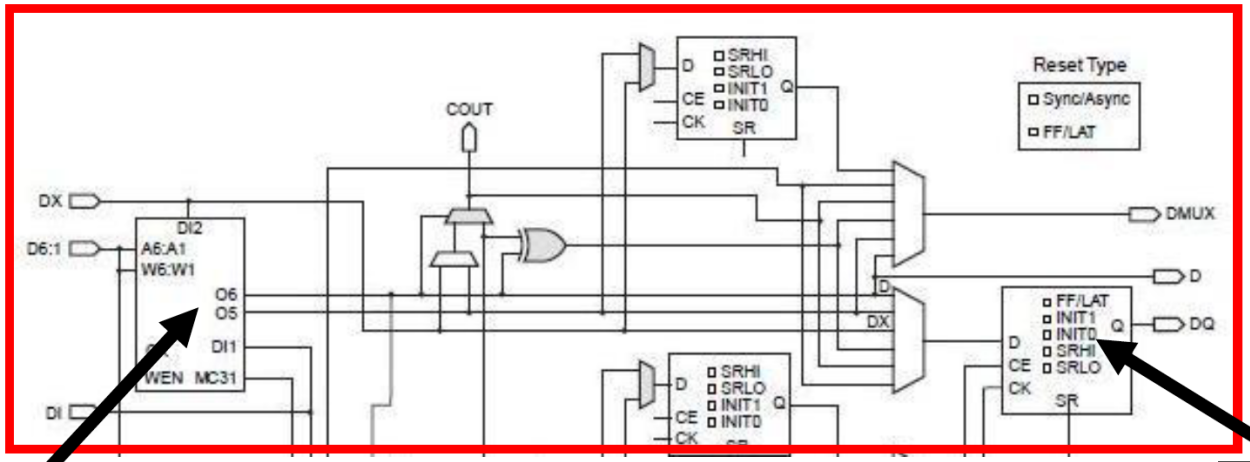
Sliceの図

7シリーズでは4組の
LUT+DFF
で1つのSliceを構成

From **Xilinx "7 Series FPGAs CLB User Guide UG474 (v1.0)** " March 1, 2011

Sliceの構造(7 series)

Sliceの一部



LUT

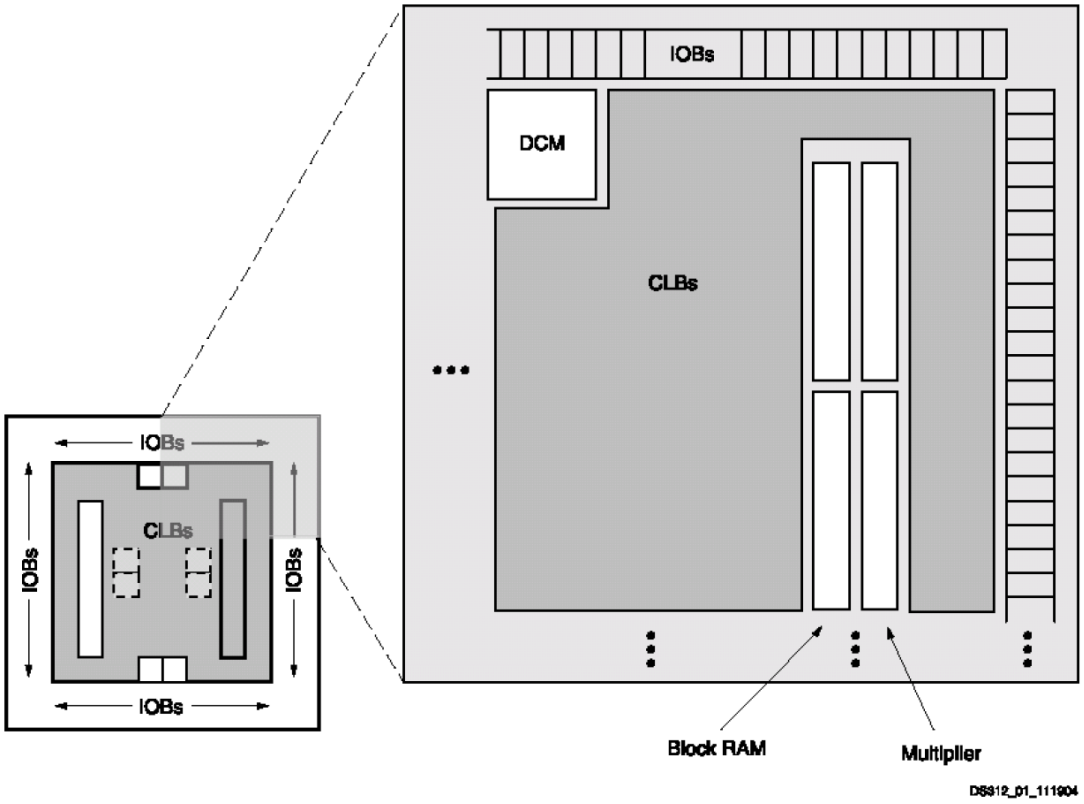
From **Xilinx "7 Series FPGAs CLB User Guide UG474 (v1.0)** " March 1, 2011

DFF

LUT以外の機能（例えばシフトレジスタやメモリ）としても使用可

実際はD以外にもなる

FPGA全体の構成



Notes:

1. The XC3S1200E and XC3S1600E have two additional DCMs on both the left and right sides as indicated by the dashed lines. The XC3S100E has only one DCM at the top and one at the bottom.

Figure 1: **Spartan-3E Family Architecture**

FPGAの構成ブロック

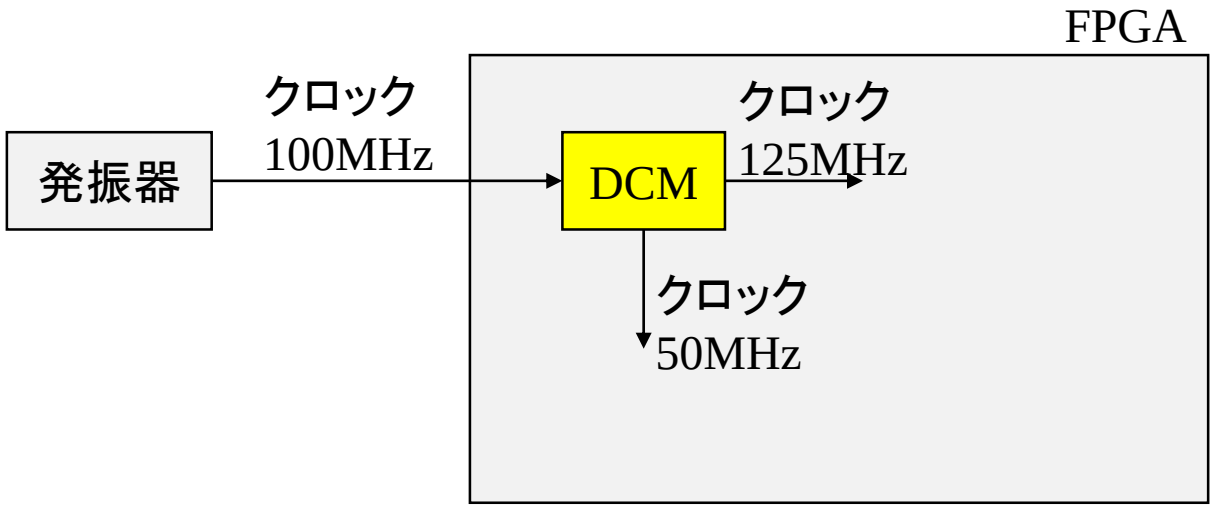
- DCM
- CLB
- IOB
- Block RAM
- Multiplier

専用回路も用意されています

データシートを参照してください

DCM（Digital Clock Manager）

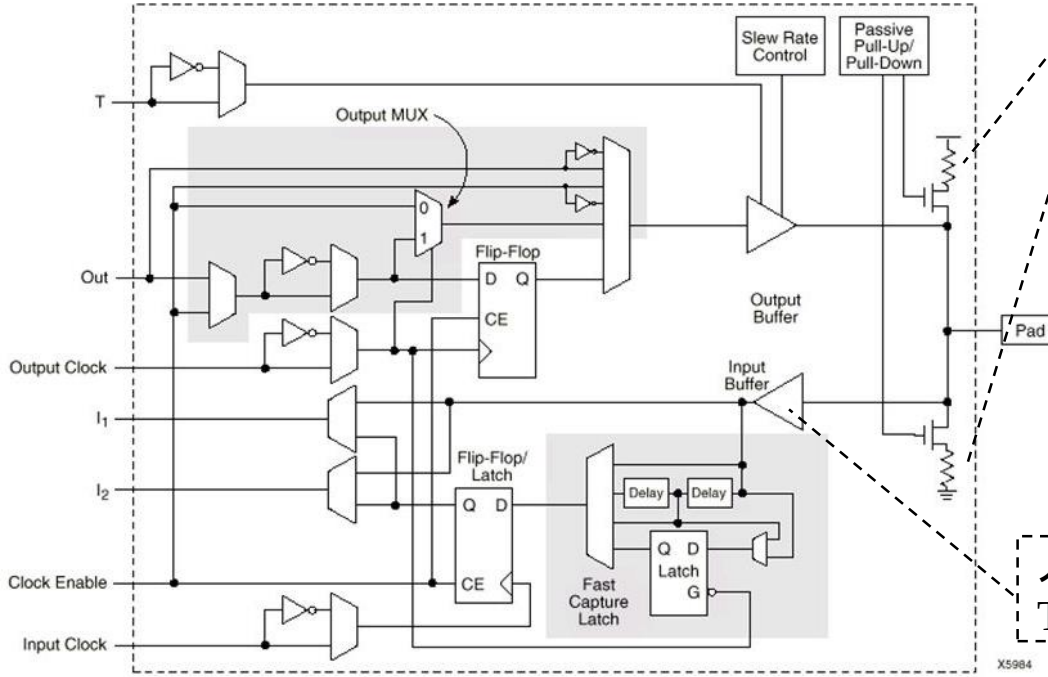
- クロック生成機能
  - ある周波数を持つクロックから別の周波数を持つクロックを作りだす
    （入力クロックのN/M倍のクロック生成可）
  - 位相差もつけることができる
  - 現在の回路は異なる周波数の複数クロックを使用する事が多い
    （例） ADCは50MHz, イーサネットは125MHzなど



DCMの使用例
100MHzのクロックから50MHzと125MHzのクロックを作り出す

IOB（Input Output Block）

FPGAのI/Oピンと論理ブロックをインターフェースする入出力専用モジュール



プルダウン抵抗、プルアップ抵抗

FPGA外部の信号

入力バッファ
TTLまたはCMOSのしきい値を持つ

Xilinx XC4000のIOB

FPGA　コンフィグレーション

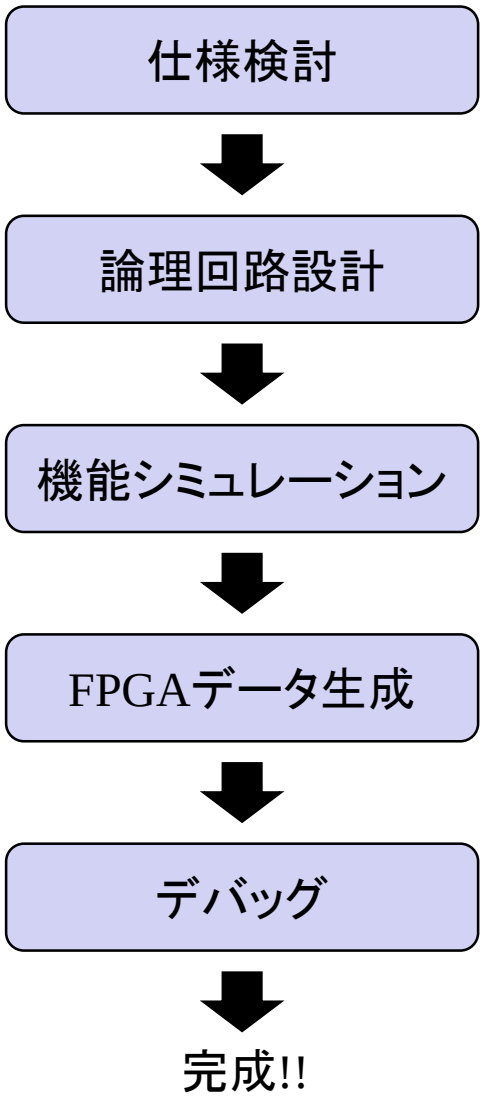
- FPGA上に回路を構成することをコンフィグレーションすると言う
- 回路情報（コンフィグレーションデータ）をFPGAに送り、FPGA内のSRAMに記憶させることでコンフィグレーションが可能
- コンフィグレーションデータ
  - 論理ブロックのデータ
  - スイッチブロックやコネクションブロックのデータ
- SRAMは揮発性なので電源を切ると、データは消えてしまう
  - 外部に不揮発性のメモリを用意し、電源投入時にそこから自動でダウンロードすることが可能

まとめ:1. FPGAの仕組み

- 基本構造は論理ブロックと配線要素(配線、スイッチブロック、コネクションブロック)
- 論理ブロックの基本はSRAM + DFF
- スイッチブロック、コネクションブロックの中の配線もトランジスタのON/OFFで接続を決める(コンフィグレーションメモリで決める)ことが可能
- 回路情報(コンフィグレーションデータ)をFPGAに送り、FPGA内のSRAMに記憶させることでコンフィグレーションが可能

2. FPGAファームウェア開発

開発概要



どんな信号が来て、どんな信号を作るのか？

回路を考え、ハードウェア記述言語（HDL）で記述

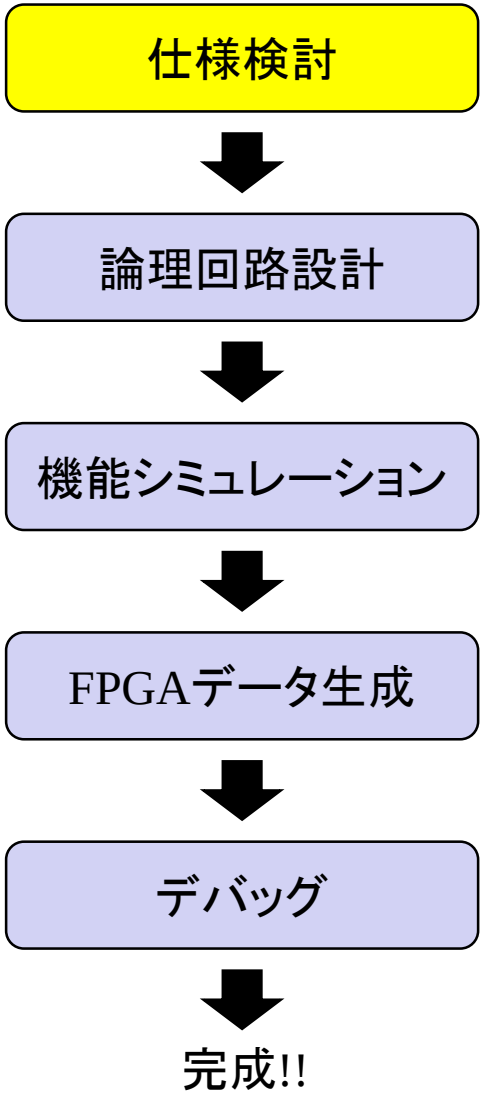
設計した通りに動作するか、コンピュータ上で確認

開発ツールを使用

実際に動かしてテスト

問題が発見されると後戻りして修正する

仕様検討



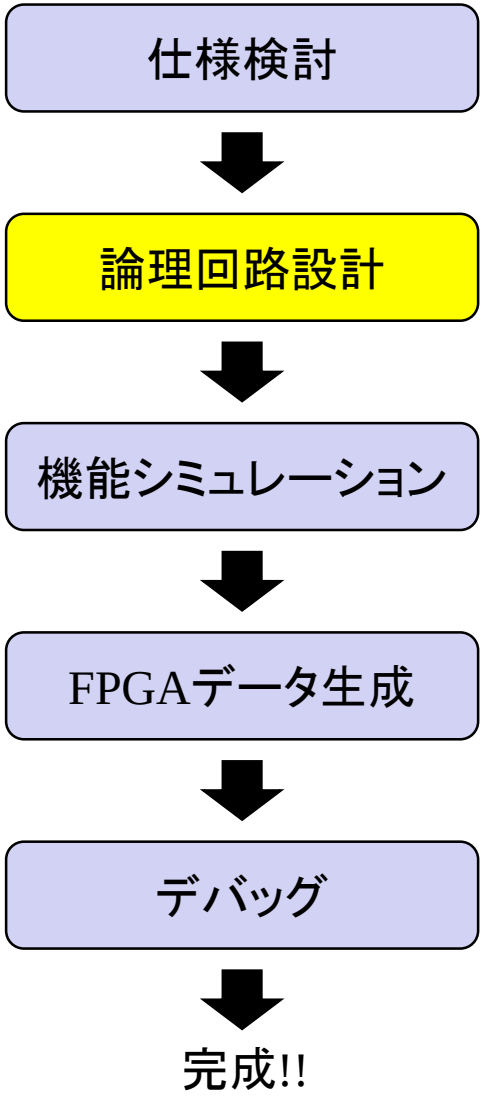
開発仕様を確定する

- 処理性能
- 使い勝手
- 形状
- I/F

スケジュールも重要!忘れないように

明確にしてから開発しないと
使えない物を作ってしまう事になる！

論理回路設計



- 通常、ハードウェア記述言語（HDL）で記述する。
- 代表的な言語
  - VHDL
  - Verilog-HDL
- 何故HDLを使うのか？
  →回路図入力には幾つかの問題がある
  - 基本的に1信号を1本の線で表現
  - 作業効率が悪い
  - 回路図が複数枚になると信号の接続先を調べるだけでも大変
  - 大規模回路では読みにくい
  - 入力ミスが多くなる

HDLの魅力

設計作業を効率よく行うために誕生

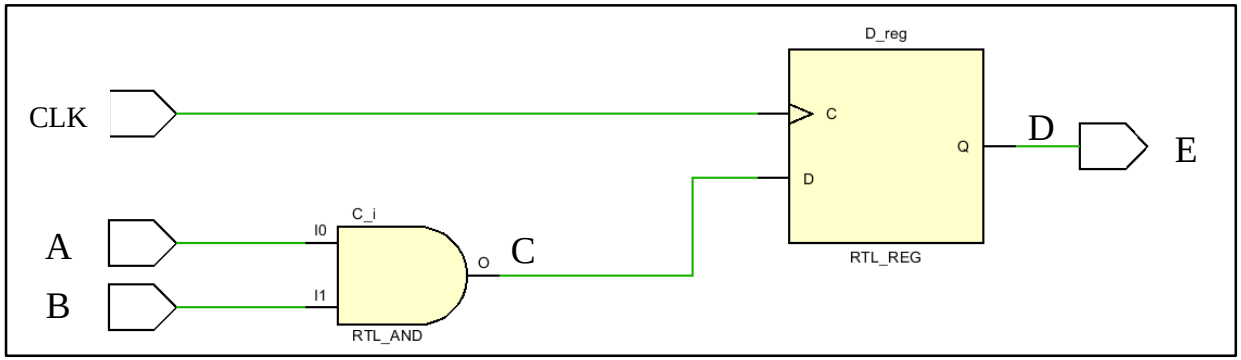
- 効率良く入力できる事で入力ミスを少なる
- 大規模回路が扱えるようになった
- 読みやすい
- コードを仕様書に近づける

※動作タイミング詳細の理解は回路図の方が優れています。
　ブロック図、回路図なども組み合わせて設計を進めてください

HDLの例（Verilog-HDL）

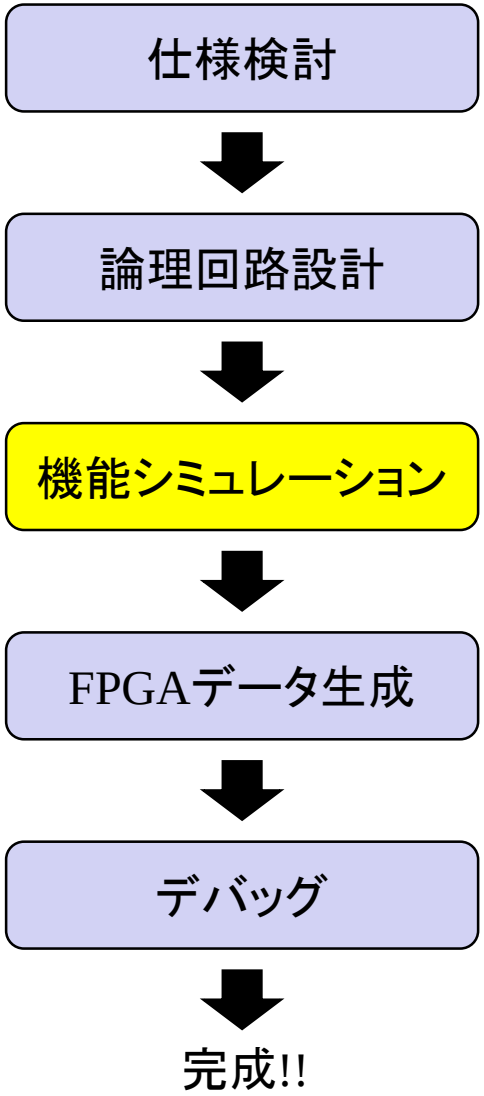
```
module TOP(
        input A,
        input B,
        output E,
        input CLK
    );


    wire C = A & B;
    reg D;
    always@(posedge CLK) begin
        D <= C;
    end

    assign E = D;

endmodule
```

スタイルはC言語に似ている



Verilog-HDLによる回路の例

機能シミュレーション



- 設計した回路が思ったように動くか検証する
  希望する動作でない時はコードを修正する
- 純粋に論理動作をシミュレーション
- 伝播遅延などアナログ量は無視
- 同期回路で設計する場合、ほとんどの部分は論理シミュレーションでよい

機能シミュレーション

```
always begin
    CLK = 0; #(5);
    CLK = 1; #(5);
end

initial begin
    A = 0;
    B = 0;
    #(40);
    A = 1;
    #(30);
    B= 1;
    #(20);
    A = 0;
end
```

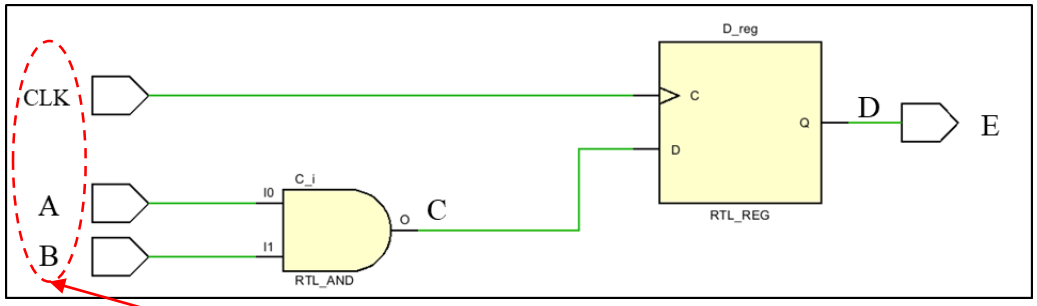
クロック

初期状態

40ns後にAが1になる

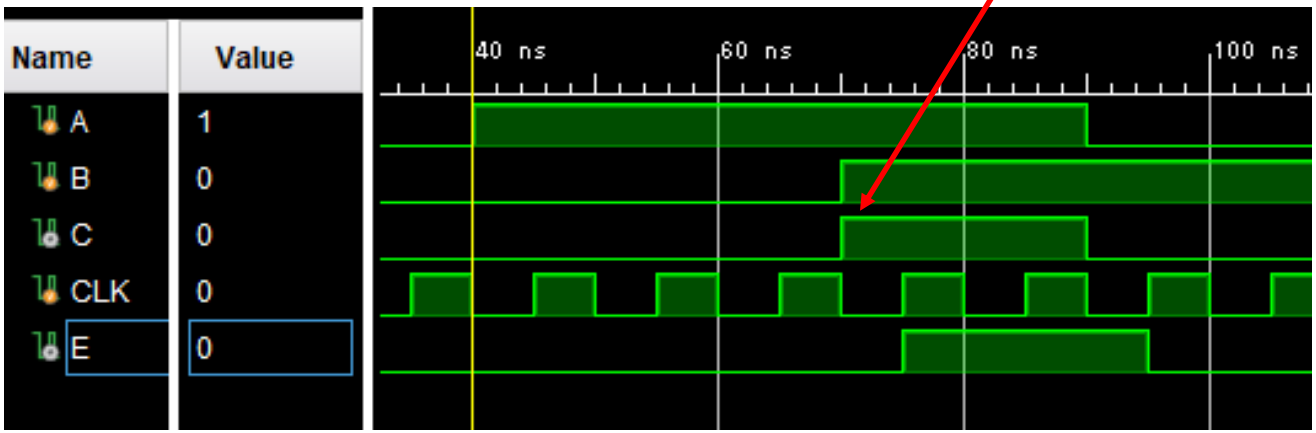
その30ns後にAが1になる

テストベンチの記述の例



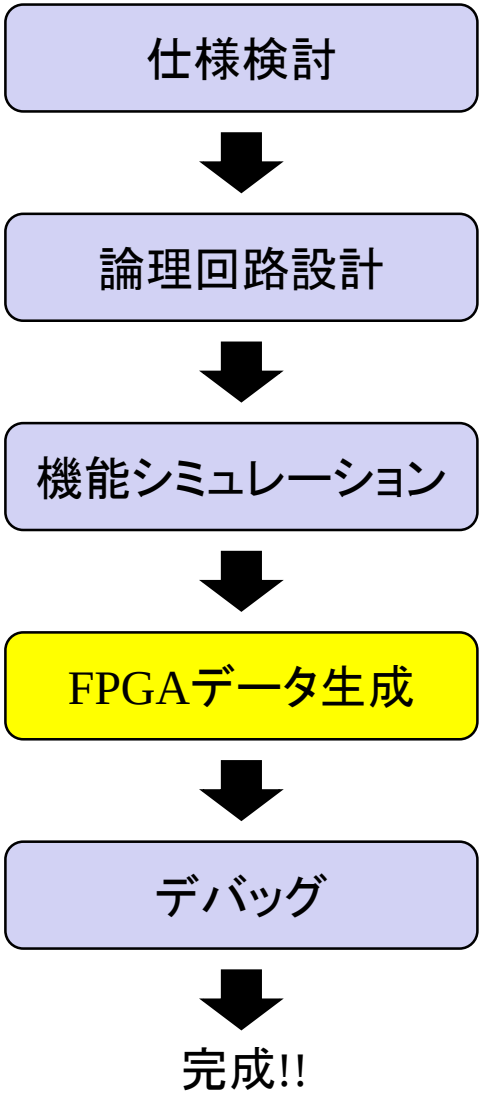
信号を与える

遅延なく値が変化



シミュレーション結果

FPGAデータ生成



- FPGAデータ生成は、基本的には開発ツールが行う
- 大きく分けて3つの工程がある

FPGAデータ生成 – ①論理合成 –

仕様検討

↓

論理回路設計

↓

機能シミュレーション

↓

FPGAデータ生成

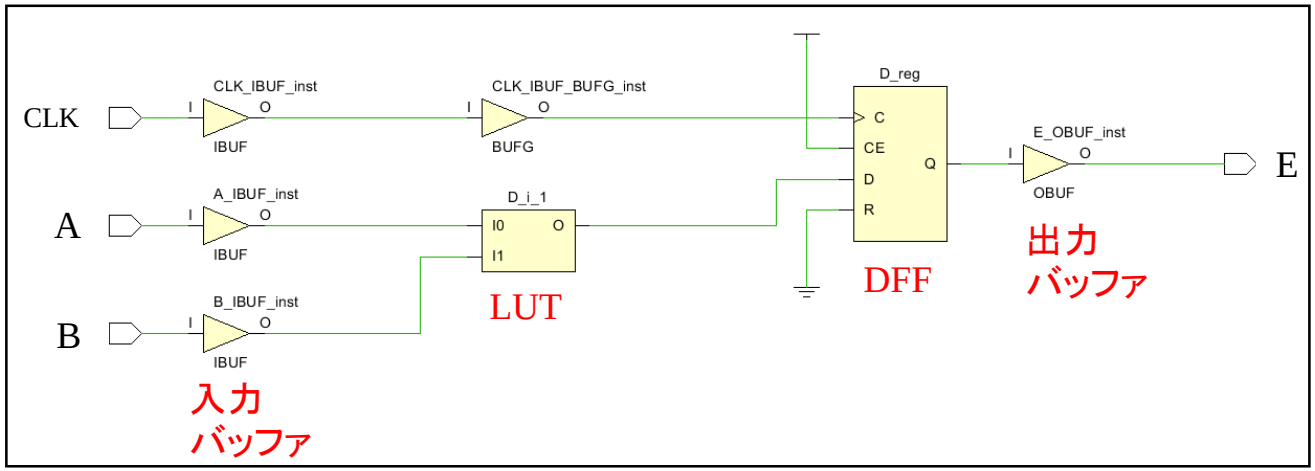
↓

デバッグ

↓

完成!!

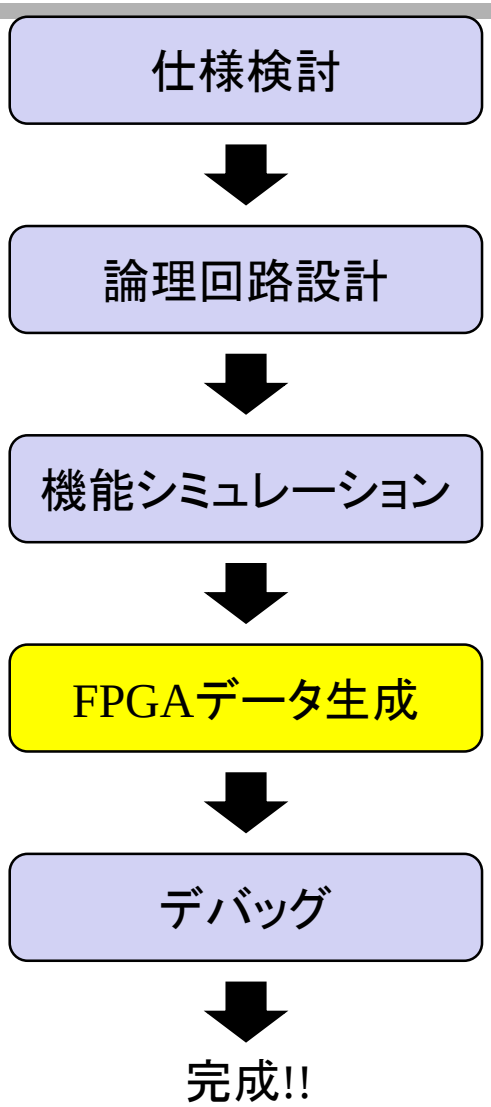
①論理合成
HDLで記述された回路を実際のゲート回路（ネットリスト）に変換



論理合成で作成したネットリストの例

LUTやDFF等の回路部品で構成されている

FPGAデータ生成 – ②配置配線 –



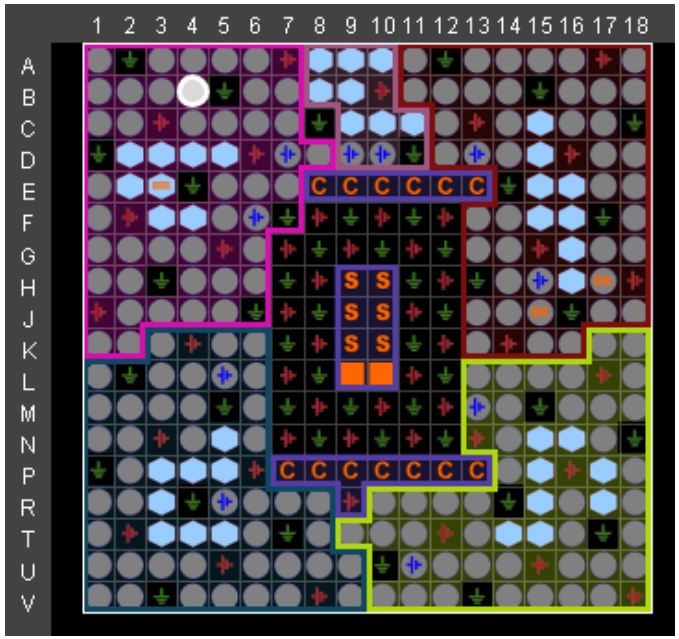
②配置配線
論理合成で作成したネットリストをチップ上の論理ブロックや配線要素等に割り当てる

- ユーザーが回路に課す制約が必要

  代表的な制約
  - I/O制約
    最上位モジュールのポート信号とFPGAのピン番号の関係
  - タイミング制約
    上限動作周波数など

I/O制約

- 設計した回路の入出力信号とFPGAピン番号と対応させる
- 外部信号のIO規格、プルタイプ、終端抵抗の有無などについても指定する



Xilinx Artix-7 xc7a100tcsg324-1
におけるパッケージピン図
信号を入出力するピンが二次元で配置されている

※IO規格

チップ間通信のために様々な規格が制定

用途に合わせて選択する
多くの場合はFPGAに接続する部品の規格に合わせる

良く使われる規格は
- LVCMOS
- LVDS

FPGAデータ生成 – ②配置配線 –

仕様検討

論理回路設計

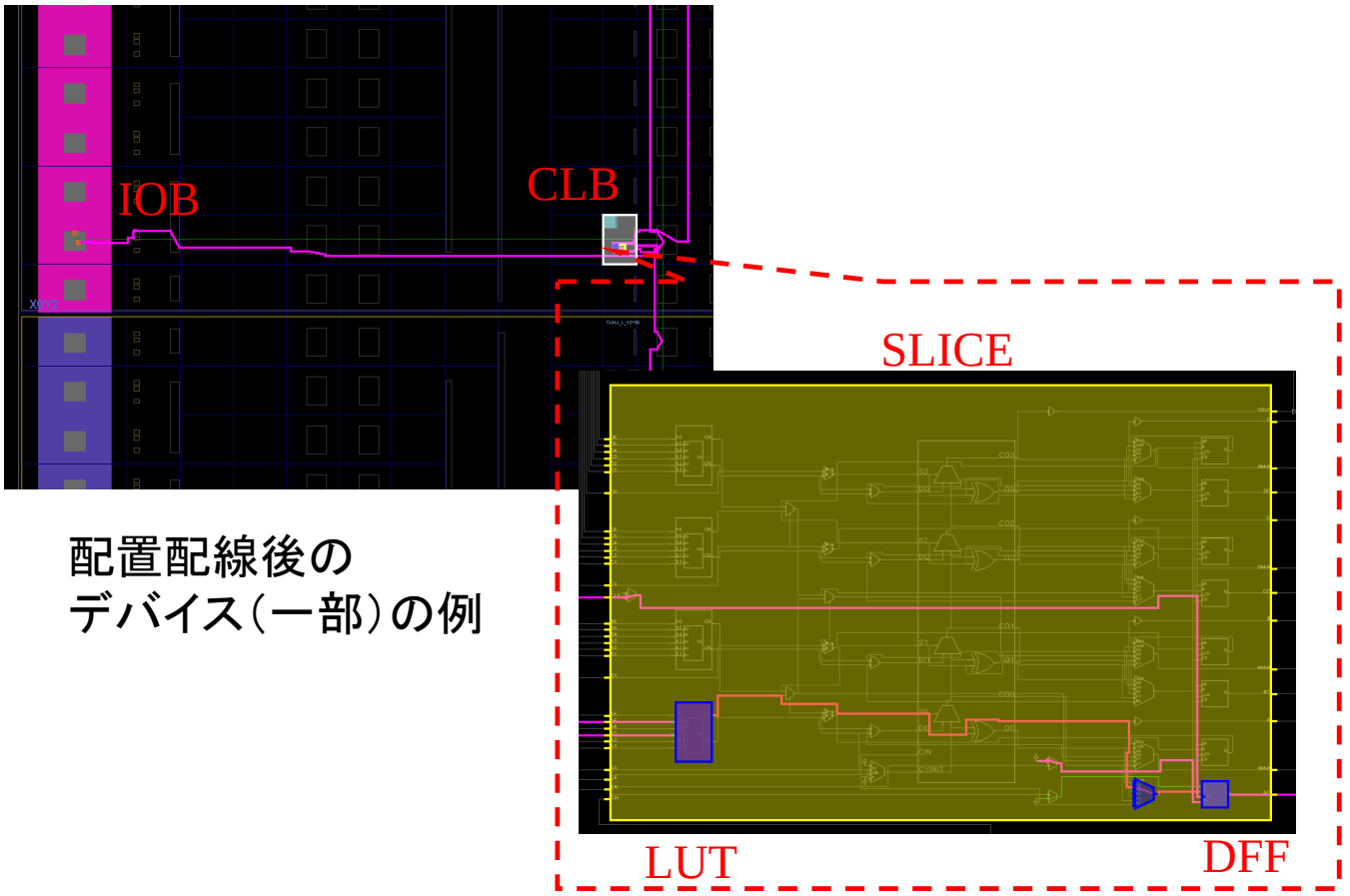
機能シミュレーション

FPGAデータ生成

デバッグ

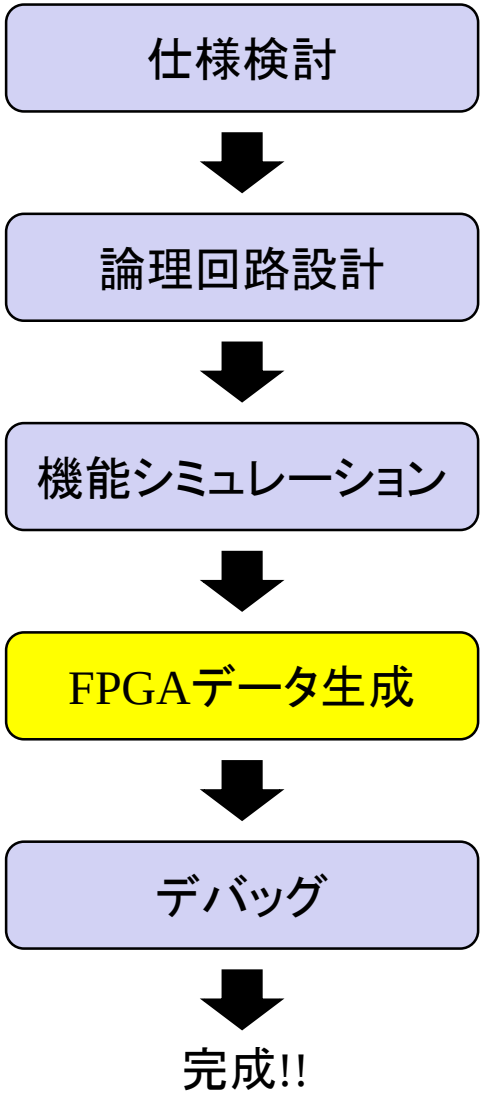
完成!!

②配置配線
論理合成で作成したネットリストをチップ上の論理ブロックや配線要素等に割り当てる



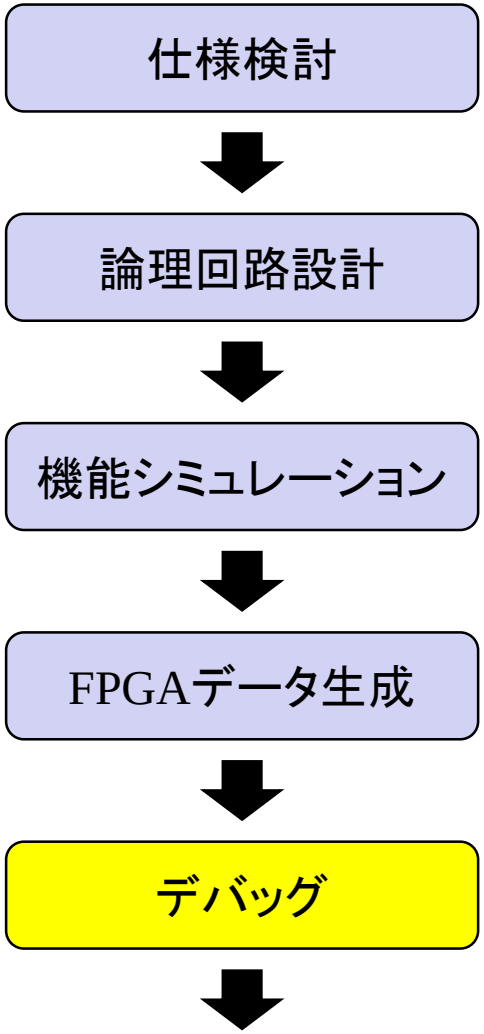
配置配線後の
デバイス（一部）の例

FPGAデータ生成 – ③データ生成 –



③データ生成
配置配線後のデータをFPGAにダウンロードするデータを生成

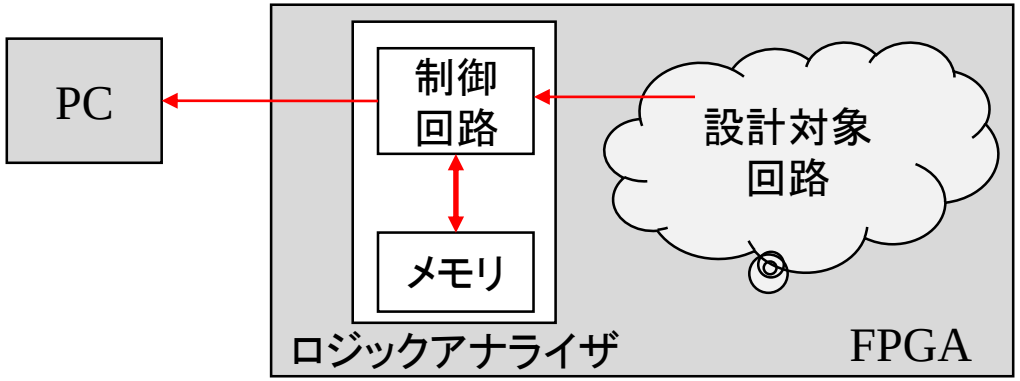
デバッグ



- 生成したデータをFPGAにダウンロード
- 実際の回路では最初は動かないことが多い
  問題点を明らかにし、思ったように動作するよう修正する
  → FPGAに組み込むことができるデバッグツールが便利

「xilinx　のデバッグツールの紹介」

- ロジックアナライザ：内部信号を観察することが可能

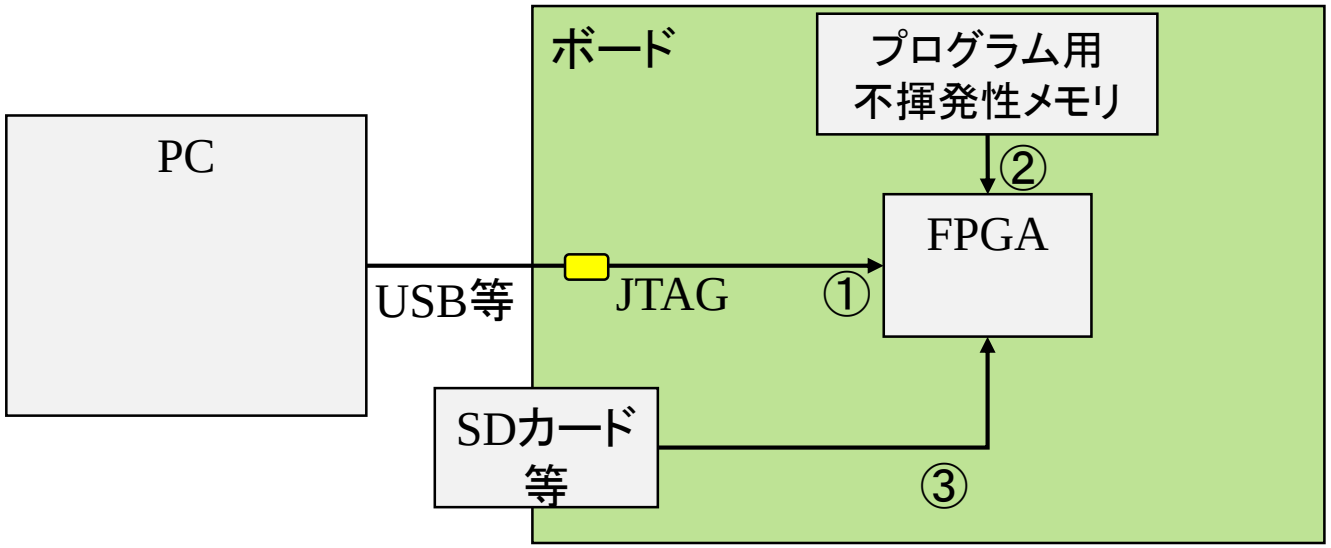


ロジックアナライザ用いたデバッグ

ダウンロードの方法

FPGAにダウンロードする代表的な方法

1. JTAG（Joint Test Action Group）を利用
   簡単な方法である反面、FPGAの電源オフでデータが消えてしまう
2. 不揮発性メモリを利用
   電源オンまたはリセット時に自動的にFPGAにデータをダウンロード
3. SDカードやUSBメモリ等を利用
   ボード上のマイコン等がSDカード等からデータを受け取り、FPGAへダウンロード



FPGAは同期回路で設計すべき

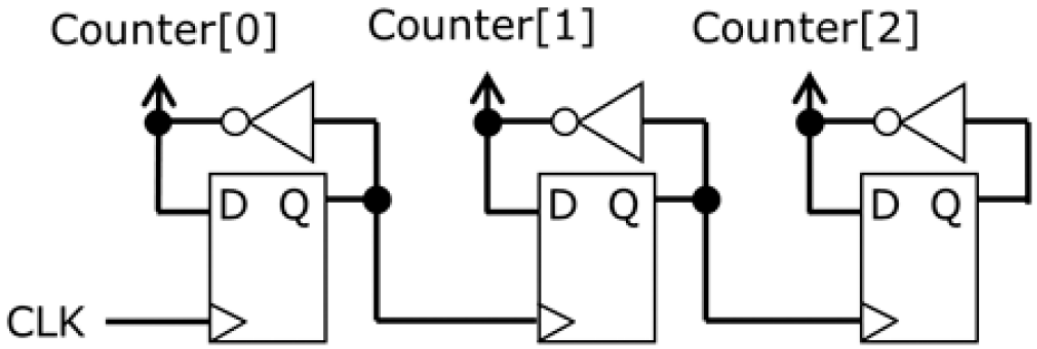
デジタル回路では同じ動きをする回路を2つの異なる手法で設計できる

- 非同期回路
- 同期回路 ➡ FPGAではこちらが主流

非同期カウンタ

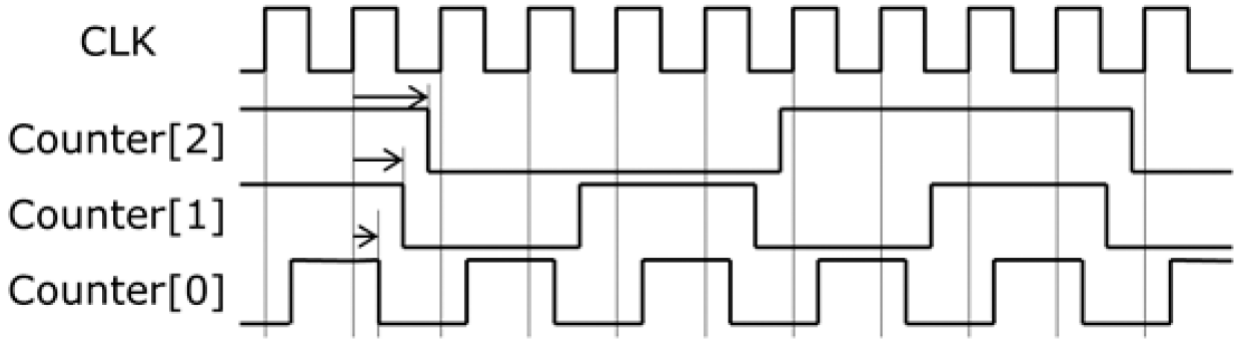
非同期回路

記憶素子（DFF）に別々のクロック信号が入力



特徴

- 回路が単純
- 遅延が伝搬していく

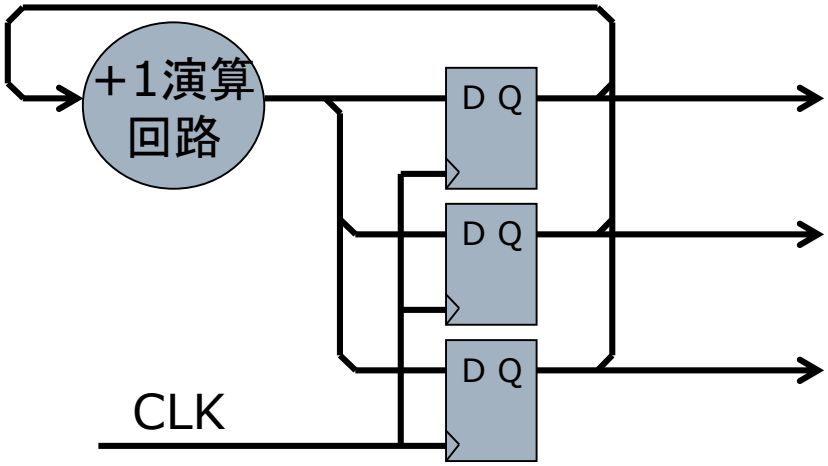


関係するゲート間の遅延を全て考慮しなければいけないため、設計が難しい

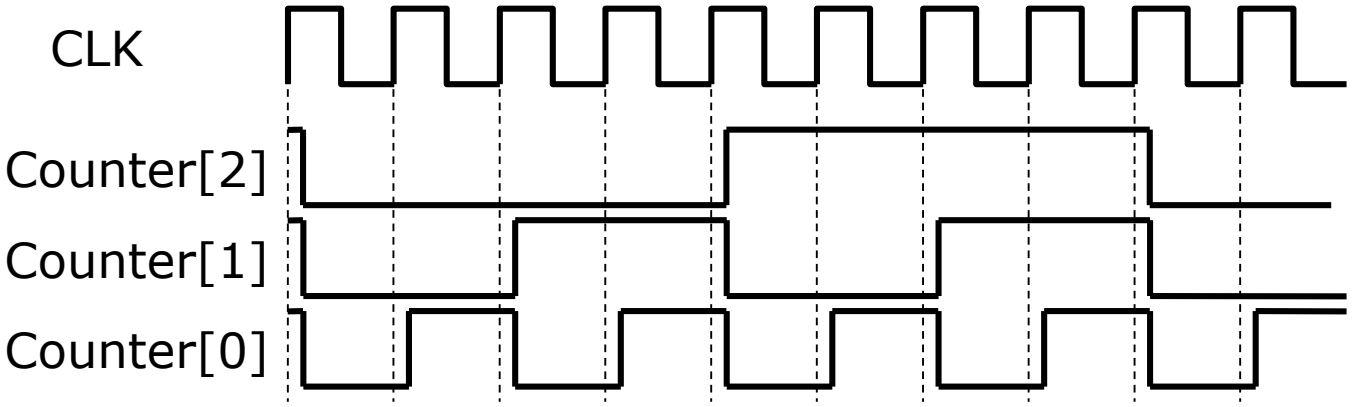
同期カウンタ

同期回路

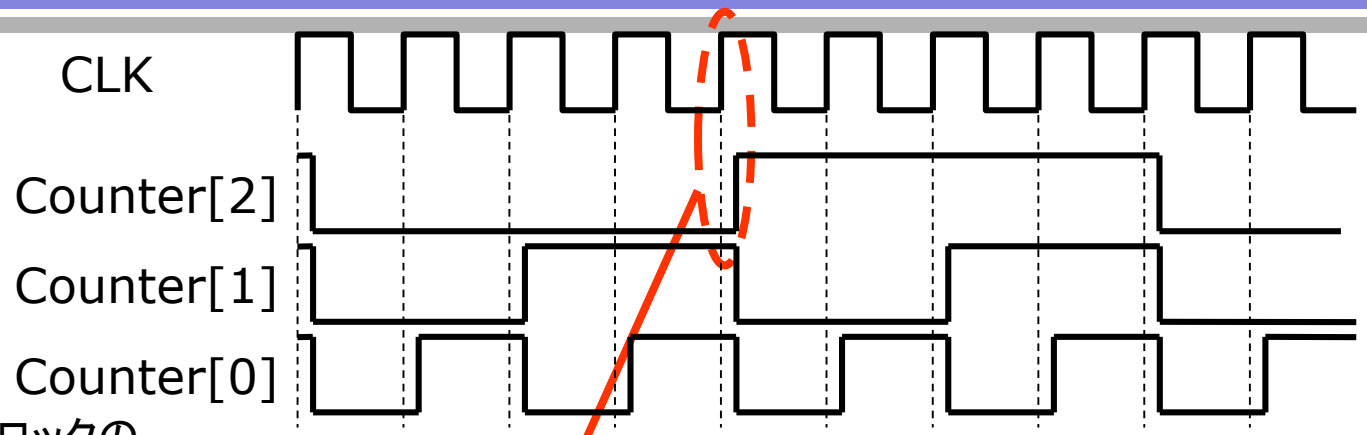
記憶素子(DFF)にシステム共通のクロック信号が入力



特徴
- 回路が複雑
- 遅延がある範囲内におさまる

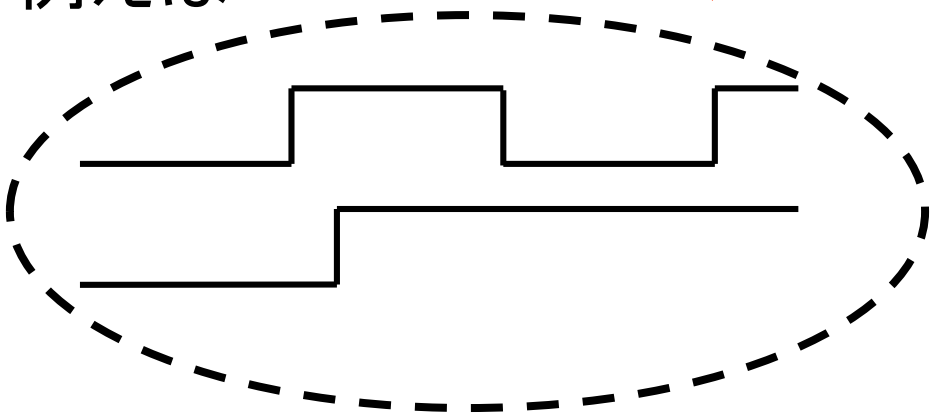


同期カウンタ



DFFはクロックの
立ち上がりでしか動作しない

例えば

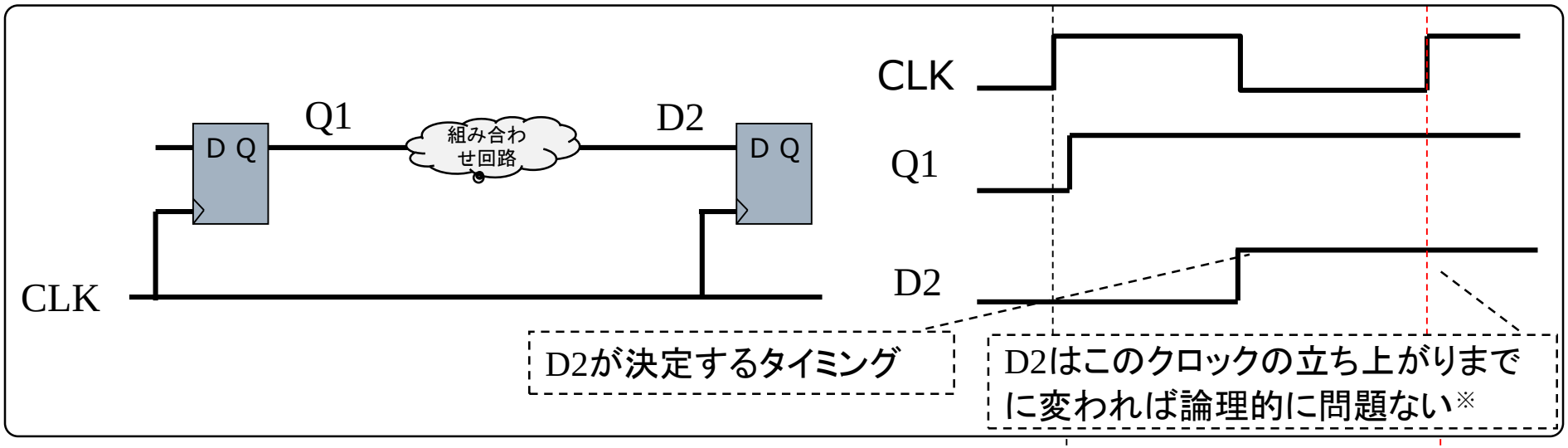


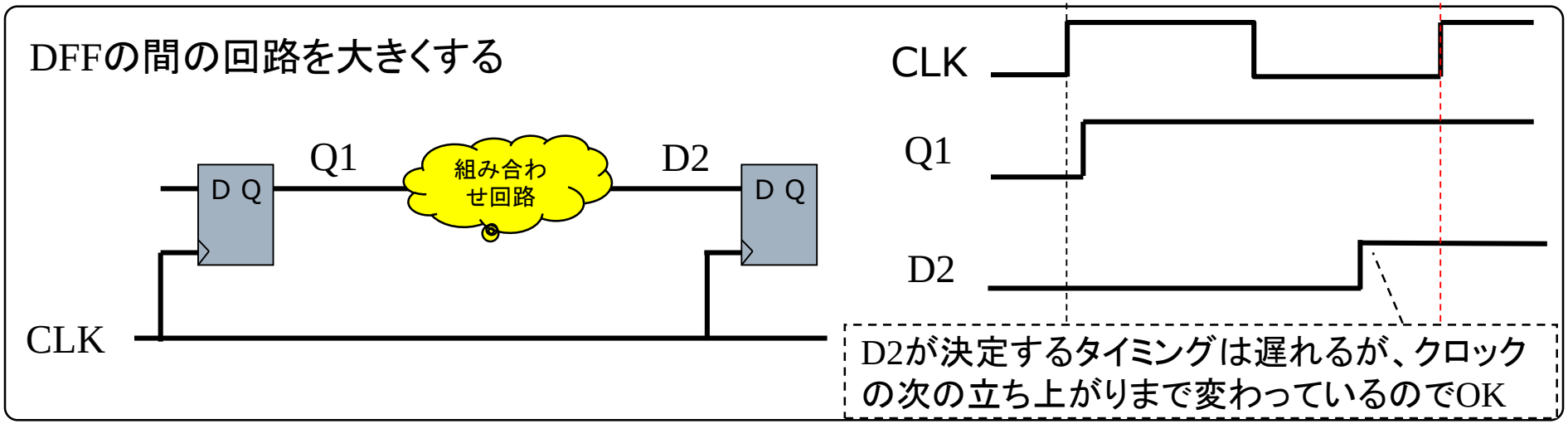
実際は値が決まるまでに時間が掛かっている

タイミング解析

- タイミング的に問題がないか解析する
- タイミング制約（タイミングに関する設計要求）を与え、その制約を満たすか計算する
- 代表的なタイミング制約は「動作周波数」
- 希望の周波数で回路が動作するかどうかを開発ツールに遅延計算させレポートさせる

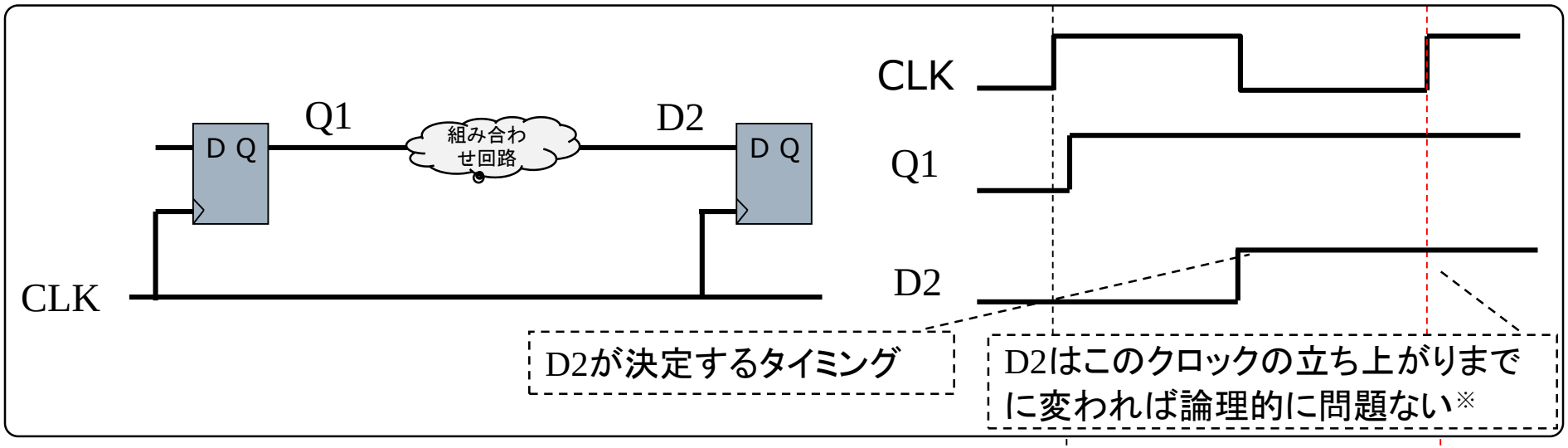
タイミング解析

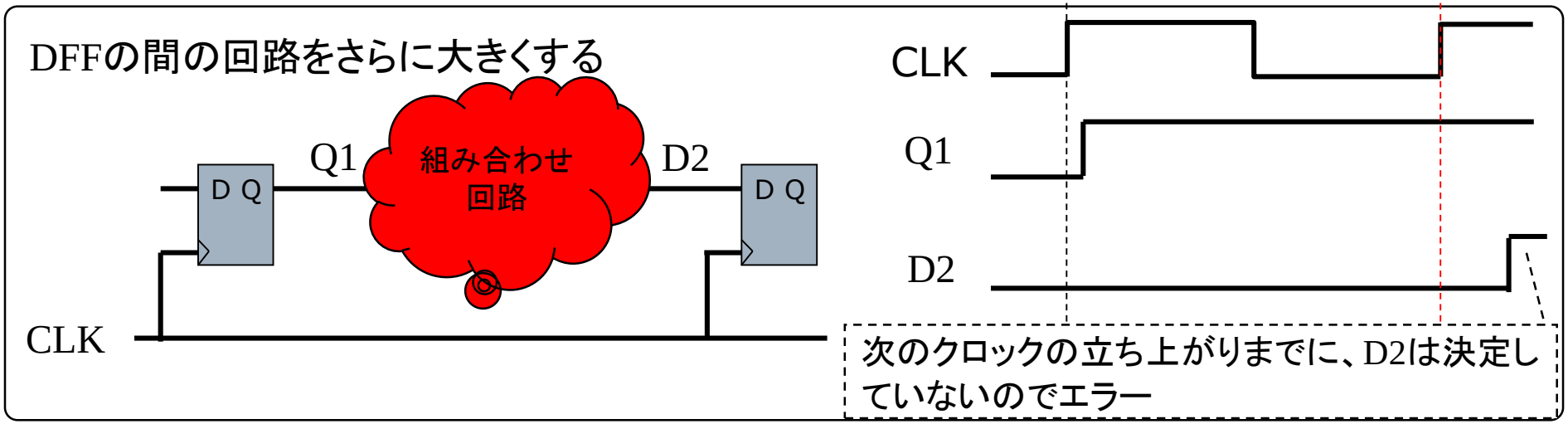




※このページでは簡単に考えるため、DFFのセットアップタイムを0と考えています

タイミング解析

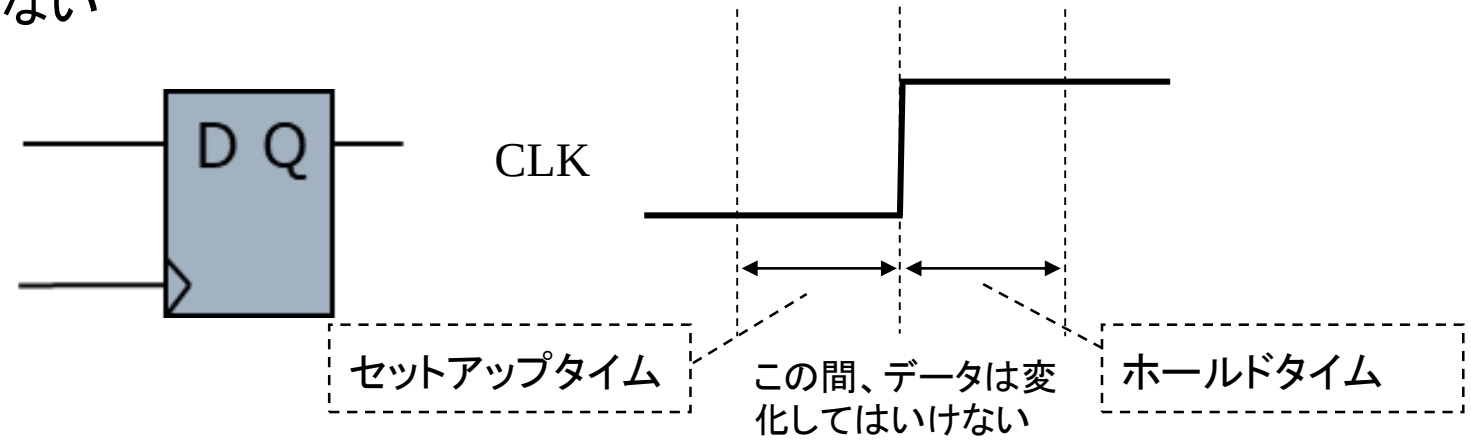


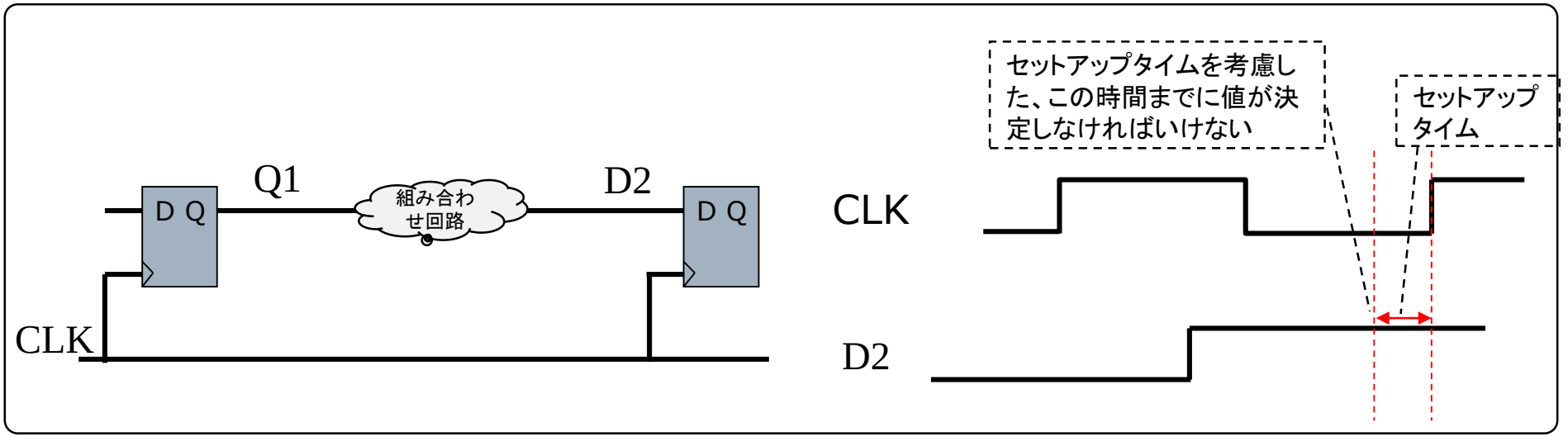


※このページでは簡単に考えるため、DFFのセットアップタイムを0と考えています

DFFにおけるセットアップタイムとホールドタイム

- DFFの入力クロック･エッジの前後の時間、入力データは安定していなければいけない





タイミング解析

開発ツールは全ての回路でタイミングに問題がないか計算してくれる

タイミング的に一番きついところでも、0.239nsの余裕がある

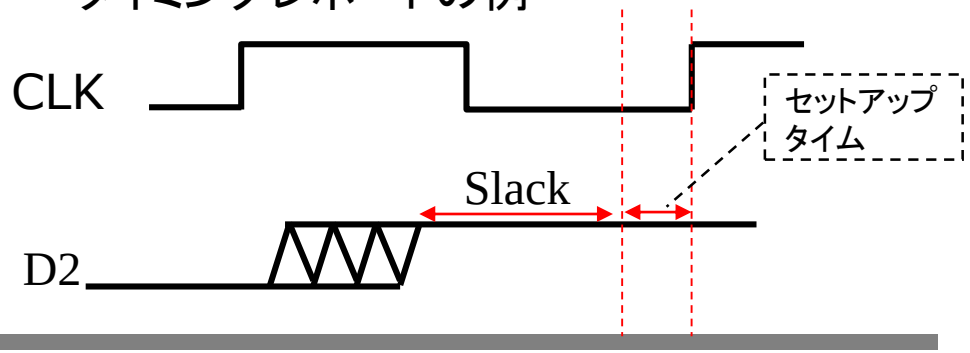
エラーなし!!

Setup		Hold	
Worst Negative Slack (WNS):	0.239 ns	Worst Hold Slack (WHS):	0.052 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	73046	Total Number of Endpoints:	73046

All user specified timing constraints are met.

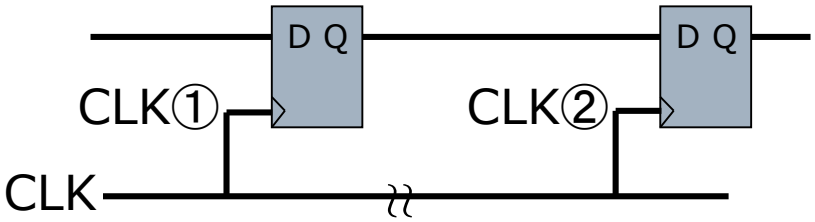
Xilinx 開発ツール Vivado上における
タイミングレポートの例

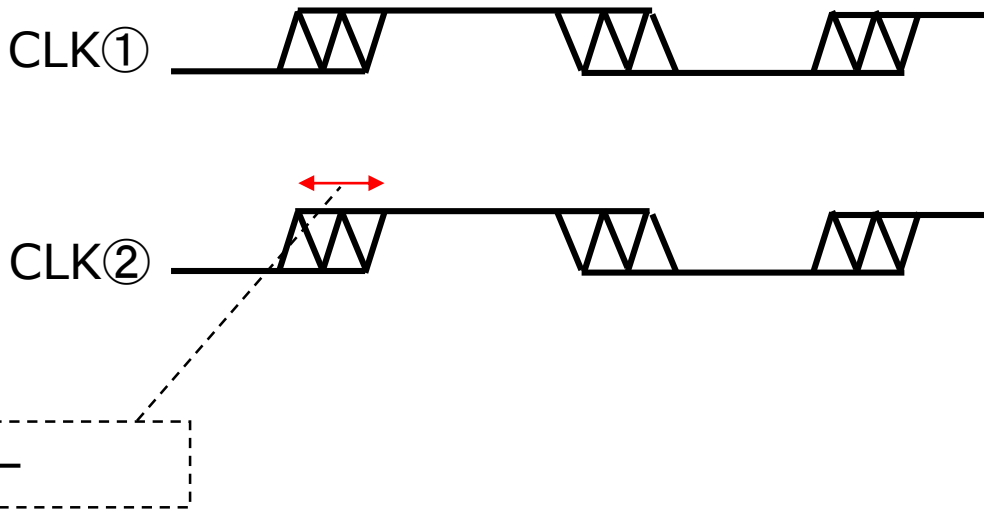
※Slack・・・タイミングマージン（余裕度）



クロックスキュー

クロックスキューとはDFFに分配されるシステム共通のクロックの時間差



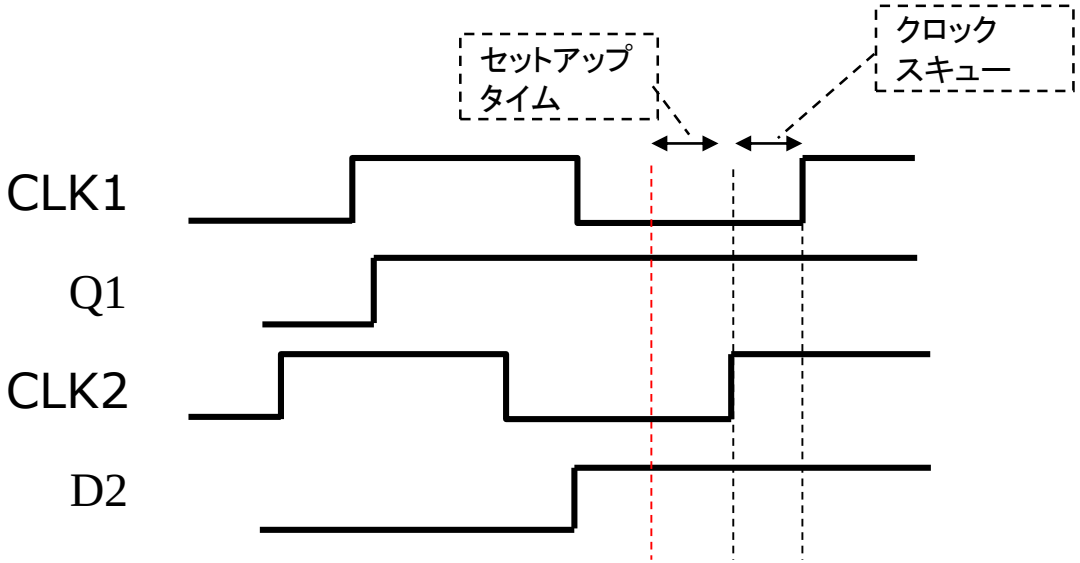


クロックスキューは小さい方が望ましい

クロックスキュー

クロックスキューが大きいと、タイミングが厳しくなる

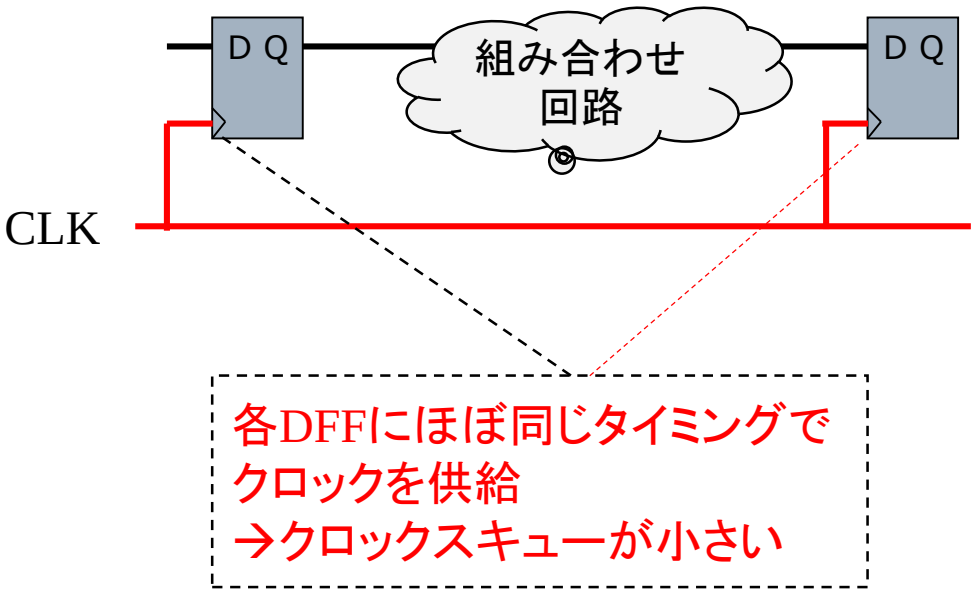
クロックスキューにより、CLK1の立ち上がりがCLK2の立ち上がりより遅い場合

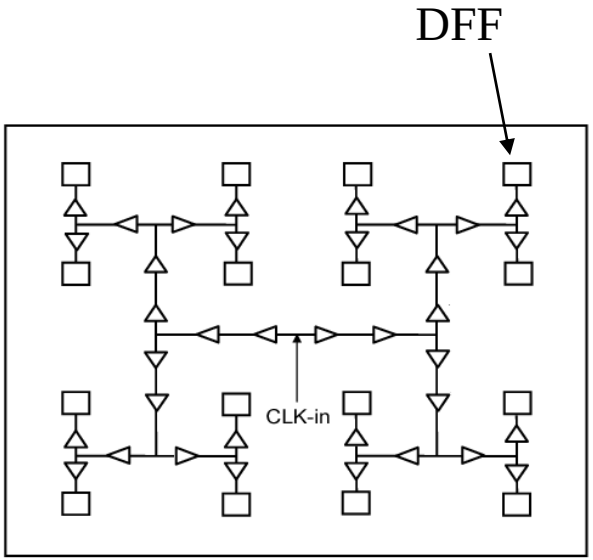


セットアップタイムとクロックスキューの時間の分、余裕が小さくなってしまう

クロック信号専用配線

FPGAでは、クロック信号を全てのDFFへ同時に
分配するクロック信号専用配線を利用できる





クロック信号専用配線のイメージ図

経路長がほとんど同じになるように
作られている

※クロック信号専用配線の数には限りがあるので注意が必要

まとめ:2. FPGAファームウェア開発

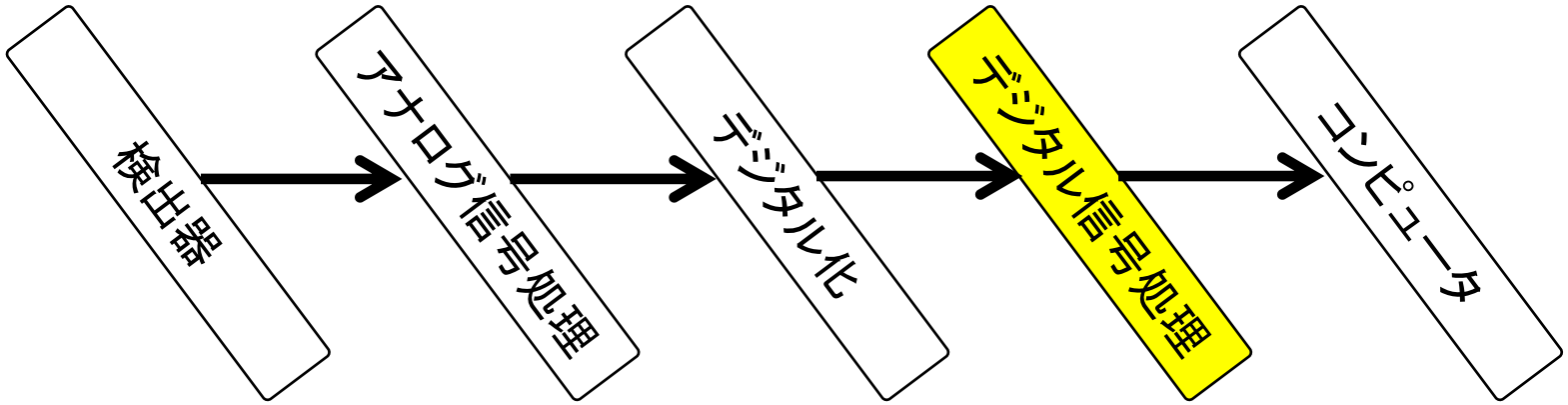
- FPGAファームウェア開発工程は、

仕様検討、論理回路設計、機能シミュレーション、
FPGAデータ生成、デバッグ

- FPGAでは非同期回路ではなく、同期回路で設計する
  - 遅延が伝搬しないため、同期回路の方が設計しやすい
  - タイミング解析を開発ツールが行ってくれる
    （タイミング制約を忘れずに!）
  - あらかじめクロック信号専用配線が用意されている

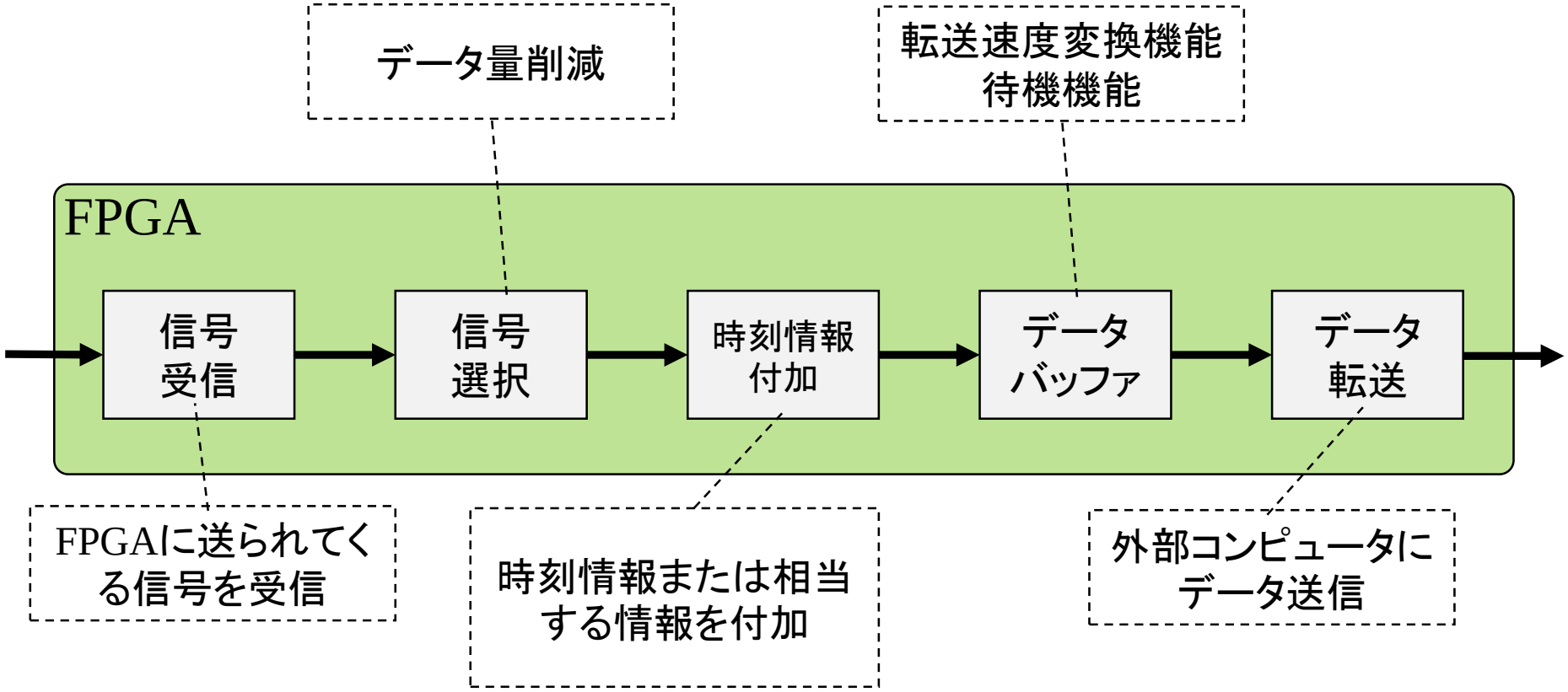
3.　FPGAを用いたDAQ

一般的なDAQシステムの構成



FPGAがよく
利用される

FPGAに実装する一般的な機能



※信号受信ブロックとデータ転送ブロック以外は順番の変更可

1. 信号受信

FPGAに送られてくる信号を受信する

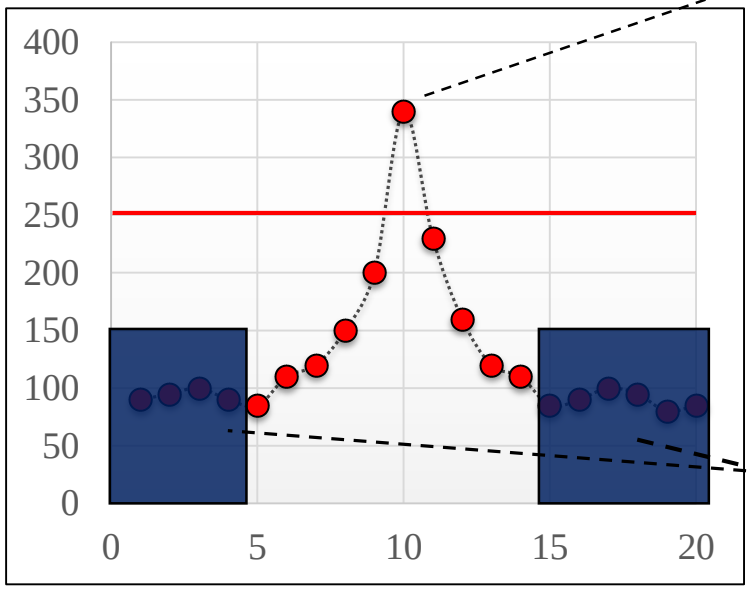
- 受信する信号の方式に合わせて回路を作成する
  - シリアル通信かパラレル通信か
  - bit数はいくつか
  - DDR（Double-Data-Rate）かSDR（Single-Data-Rate）か　等々
- データシートをよく読んで、送られてくるデータやクロックなどの信号のタイミングや仕様を理解してから回路を設計すること
- 他の回路を作る前に、ロジックアナライザ等を使って問題ないか確認するのも有効

2. 信号選択

デジタル化した検出器信号のデータを全て送れない時は
優位だと思われるデータのみ選んでPCへ送る

代表的な機能：ゼロサプレス

基準値 = 250



ここで基準値を超えた

この部分のデータ
は送らない

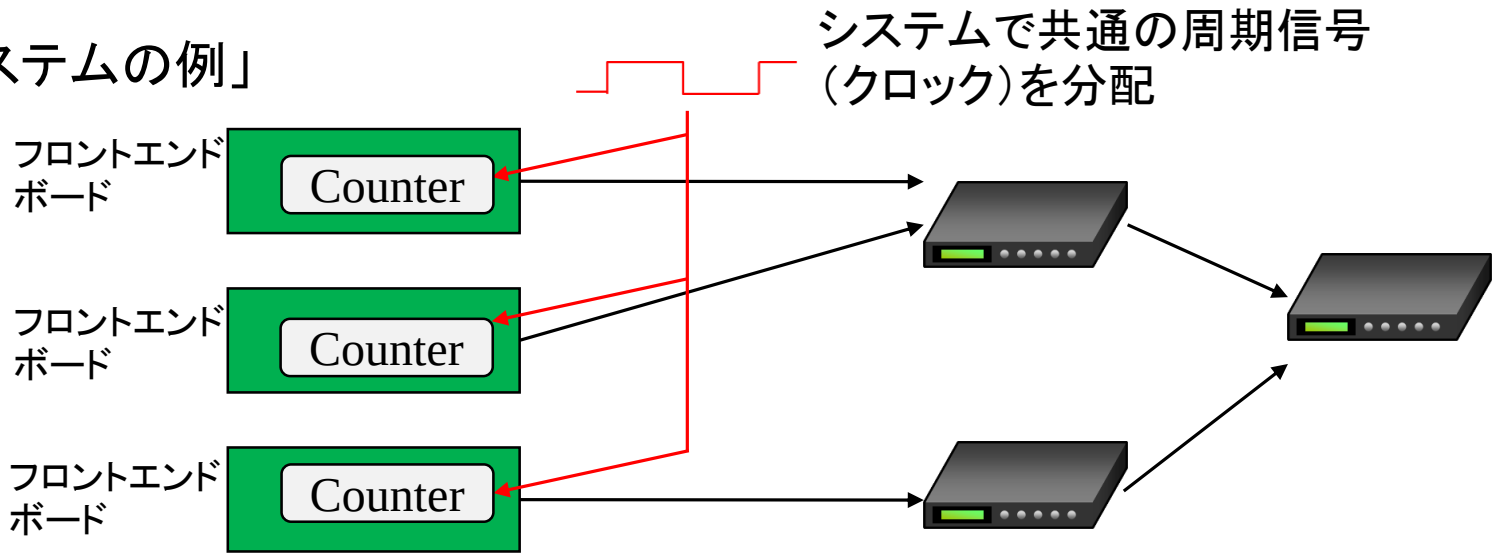
前後5単位時間のデータのみ転送

3. 時刻情報付加

異なる事象を区別するために時刻情報または相当する情報を付加する機能

- 複数の検出器で構成されるシステムの場合、複数の検出器から送られてくるデータを1つのデータとして組み立てなければいけない（イベントビルド）
- 実験で必要な時間制度、nsecからusecオーダで時刻を正確に合わせる必要がある

「システムの例」



※実際は時間原点を決める信号がさらに必要

4. データバッファ

データを一時的保存する機能

データバッファの役割

1. 転送速度変換機能
2. 待機機能

複数個のデータを一時的に保存

- Rw ≦ Rs :

データはあふれることはないので読みの手都合でデータを読み出すことができる

- Rw > Rs:

データが溢れてしまうのでデータが溢れそうになったら書き込みを止める仕組みが必要

4. データバッファ

データバッファ機能実現のための方法

1. FPGA内部メモリを使用
   - 内部メモリをFIFOやリングバッファとして利用することが多い
   - 容量は限られている

2. 外部メモリを使用
   - FPGAの外に配置したメモリを使用
   - 容量が大きいバッファが必要な場合は外部メモリが必要

※IPを使用することで回路を作りやすくなる

※IP（Intellectual Property：知的財産）とは
- ユーザーから見た場合、既に設計された回路ライブラリ
- 有償/無償で様々なIPが提供されている

5. データ転送

外部コンピュータにデータを送信するための機能

- 標準化されているI/Fを使用
  - Ethernet
  - USB
  - PCI express 等々
- 規格により、長所短所あるので適切なI/Fを選ぶこと

COMET実験ストロー飛跡用読み出し回路 ROESTIの紹介

Signal → ①Front-end → ②Waveform Digitizer → ③FPGA → ④Ethernet → Data

信号増幅、波形整形
(十分なゲインでアンプ)

高速波形サンプリング
(パイルアップ対策、
高い時間分解能)

信号処理

データ転送

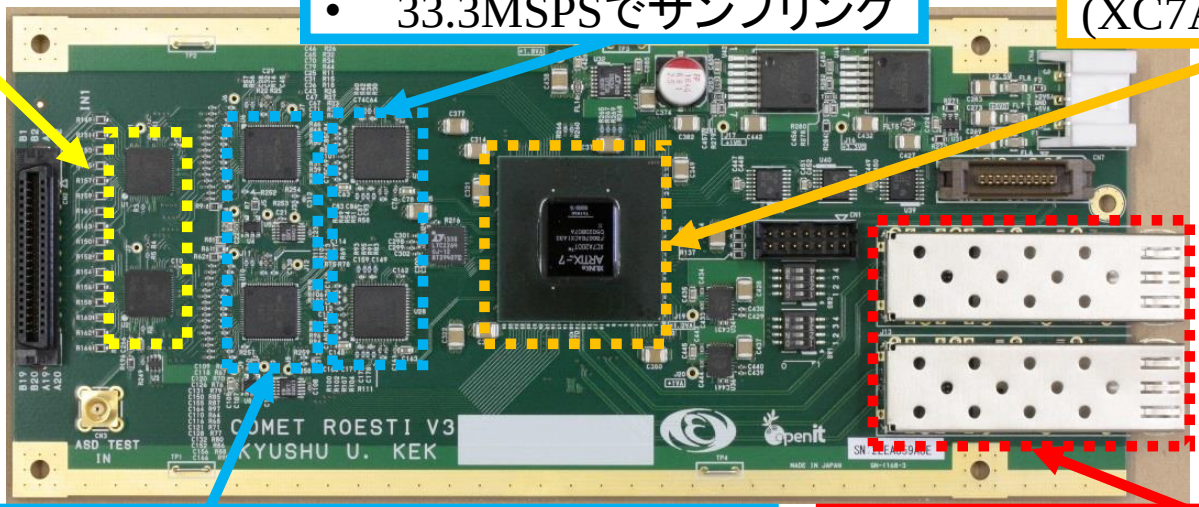
(フィードスルーでのチャンネル数の最小化)

ASD
- Gain = 1.1V/pC

ADC
- 12bit
- 33.3MSPSでサンプリング

FPGA
- Artix-7
(XC7A200T)

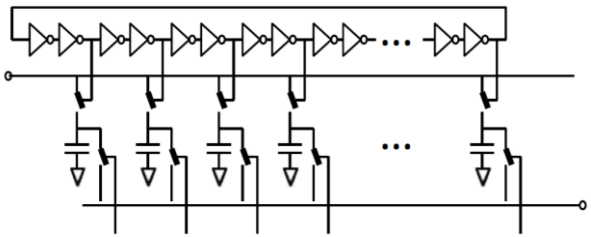
Signal
(16ch)



Data

Data

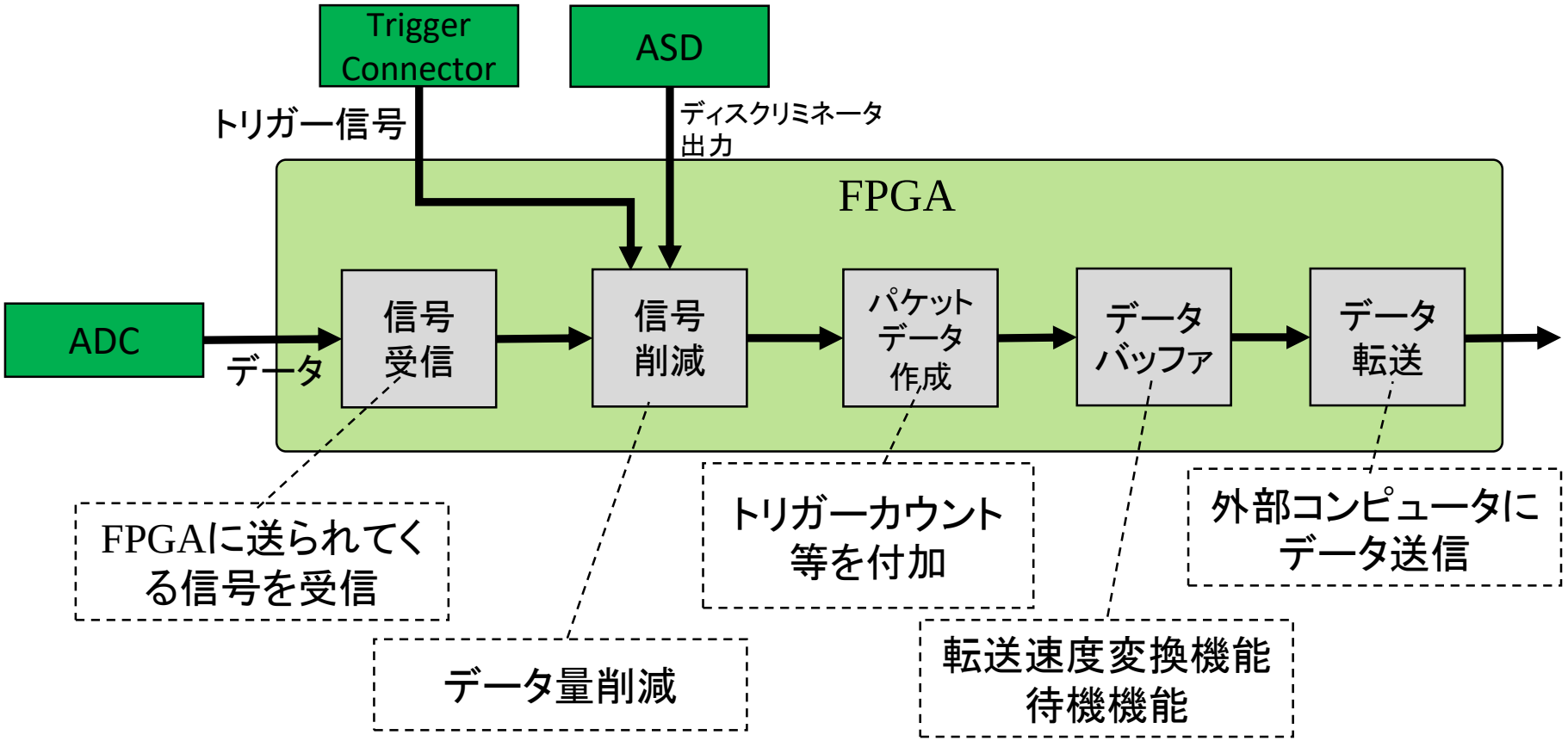
DRS4
- 1024個のスイッチドキャパシタ が接続
- 1GSPSでサンプリング



SFP connector
- 1Gbps
- デイジーチェーン

ROESTI FPGA ブロック図

一般的な機能の内容と同じような機能を備えている

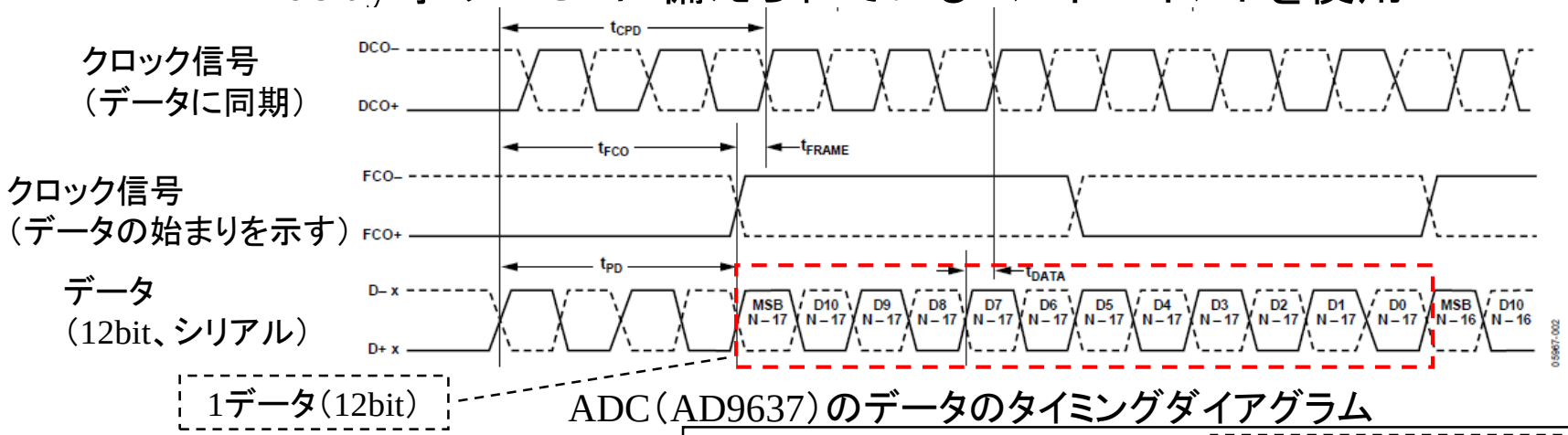


細かい説明は次のページから….

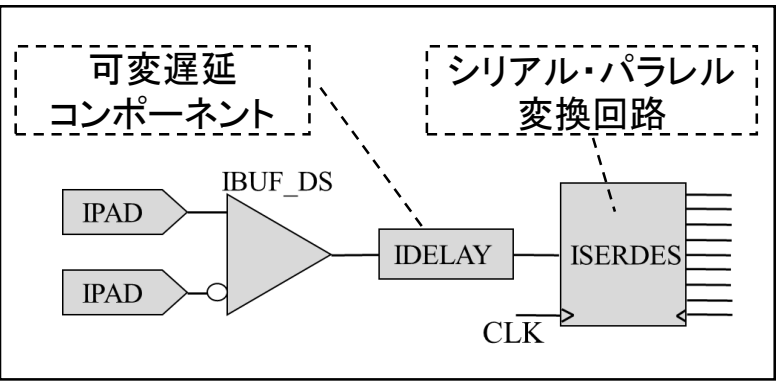
1. 信号受信

使用しているADCのタイミングダイアグラムに合わせて回路を作成

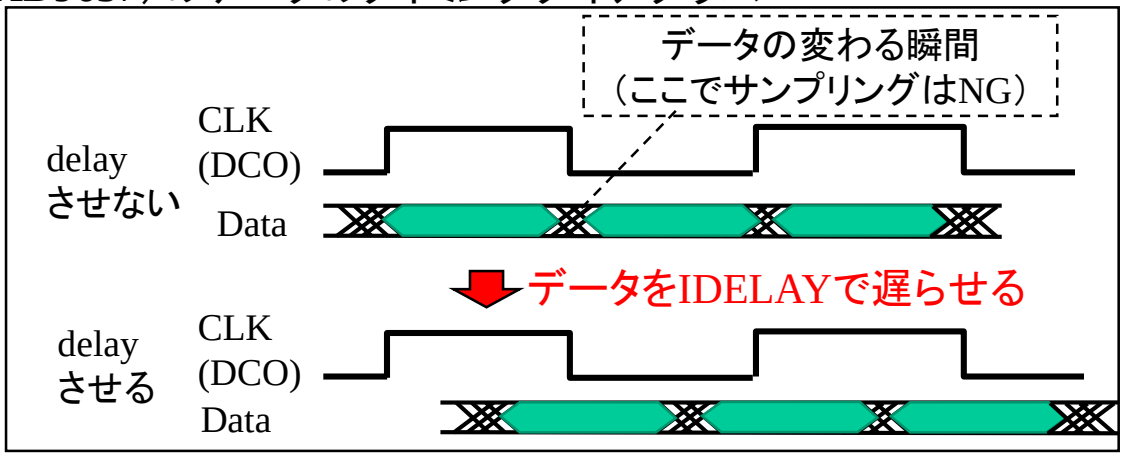
idelay等のFPGAに備えられているコンポーネントを使用



ADC（AD9637）のデータのタイミングダイアグラム



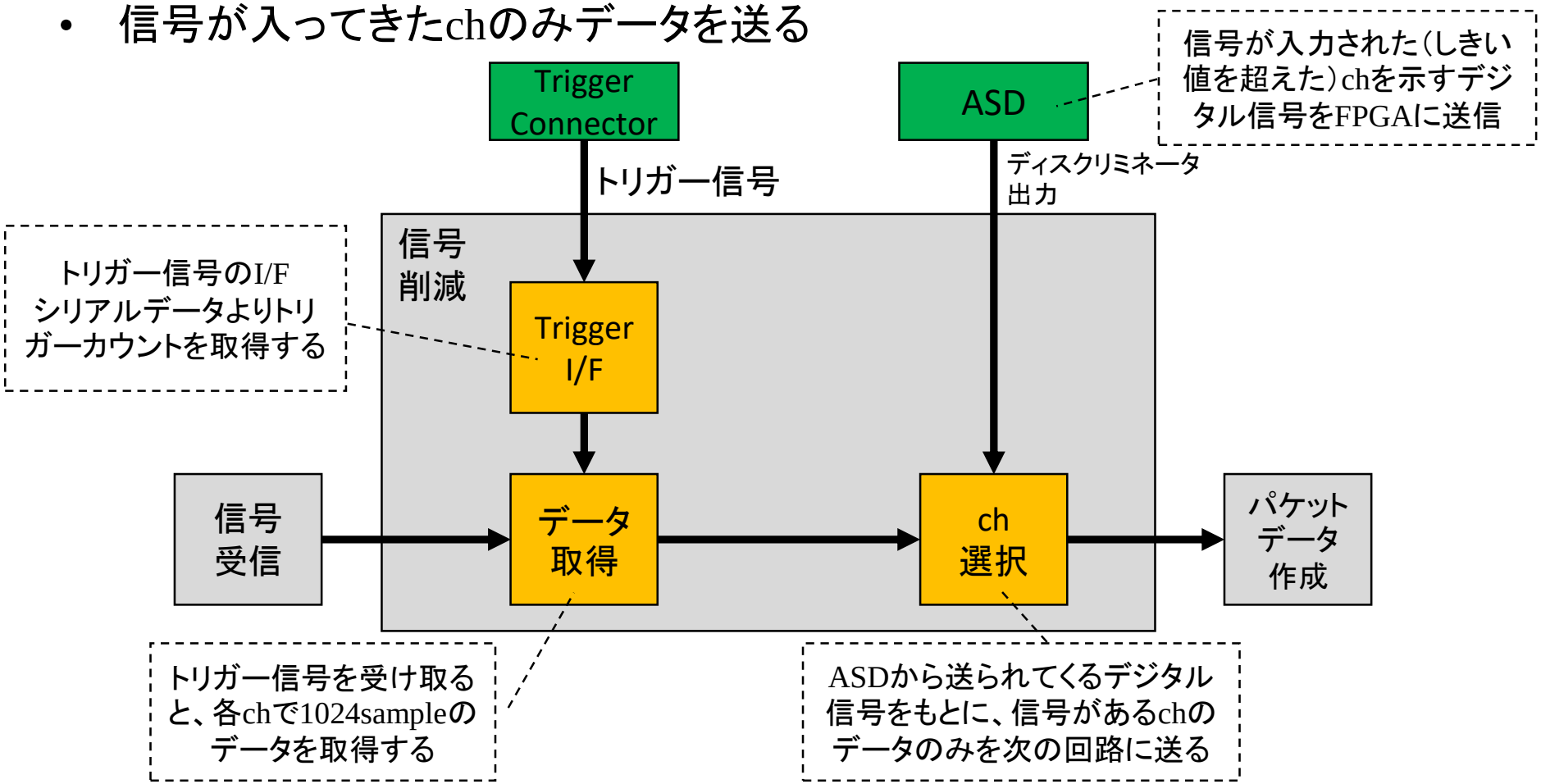
Data IF部のブロック図



ISERDESに入るCLKとデータの

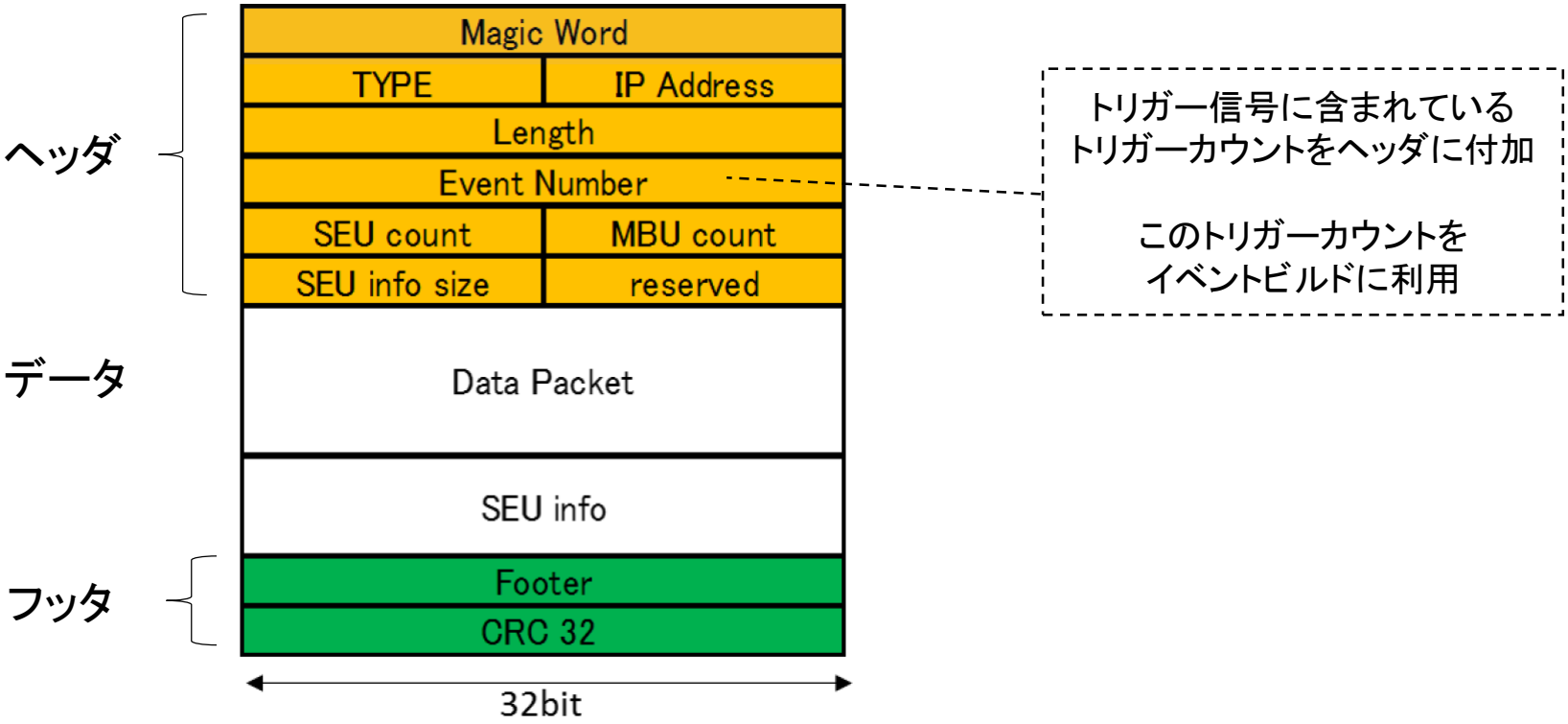
2. 信号削減

- トリガー信号を受け取ったときのみデータを送る
- 信号が入ってきたchのみデータを送る



3. パケットデータ作成

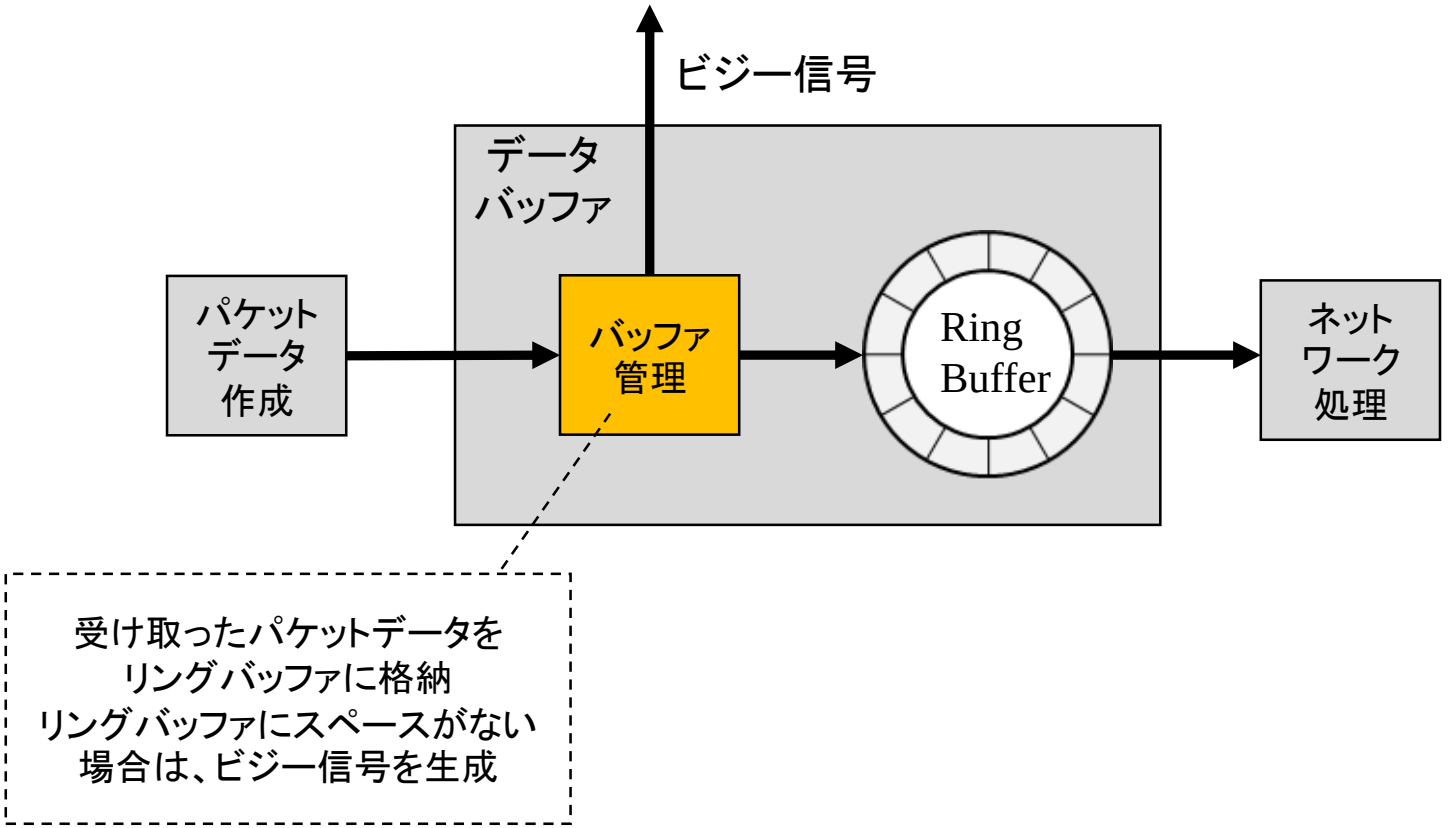
ヘッダやフッタを付加して、パケットデータを作成



データフォーマット

4. データバッファ

データを一時的にリングバッファに保存



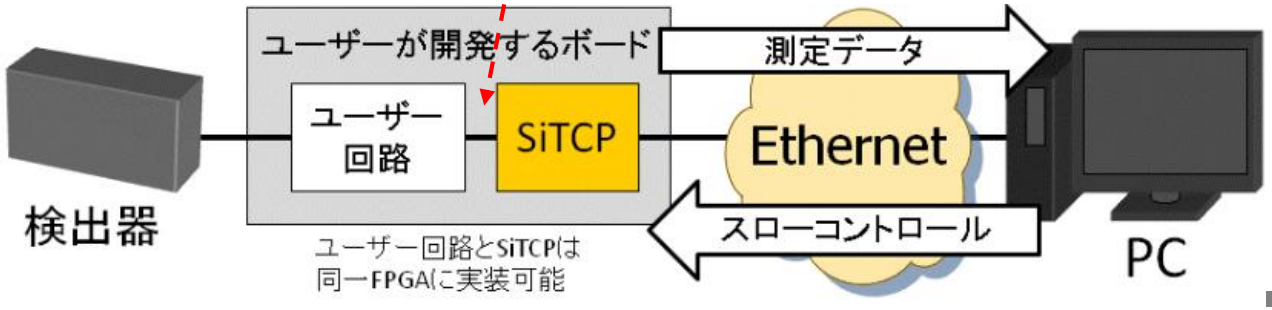
5. データ転送

データ転送にはSiTCPを利用

SiTCPとは… FPGAをEthernetに接続する技術

- ハードウエアによるTCP/IP Ethernet通信
  - 最大1Gbps の通信が可能
  - 安定したTCPデータ通信
  - UDPを用いたスローコントロール機能
- 容易な実装
  - 使いやすい容易なユーザI/F
  - 小さい回路規模
  - FPGAライブラリ(Xilinx)として配布されている

同期FIFOメモリに書き込む手順



4.　最後に

FPGA最近の傾向

様々な専用機能が搭載（予め書き込まれている）されている

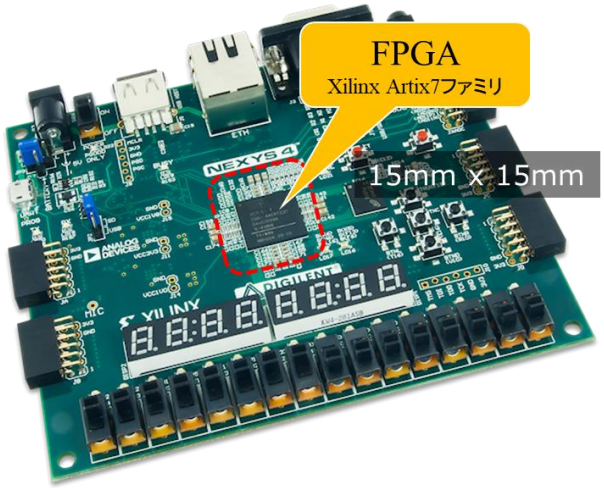
- 組み込みCPU
  - Linux動作可
  - 今はARMプロセッサが主流
- Digital Signal Processing (DSP)
  - 積和演算ユニットの事
- 高速シリアル通信
  - >10Gbps
- メモリコントローラ
  - DDR2/3 SDRAM

FPGAの習得は実践が一番

FPGAトレーニングコースをご利用ください

http://openit.kek.jp/training/2020/fpga/sokendai-ims/fpga

2020/9/3〜9/4



実習形式で手を動かしながらFPGAを
動作させ、開発ツールを使い方を学ぶ！